Cisco Crosswork Situation Manager 8.0.x Developer Guide

(*Powered by Moogsoft AIOps 8.0*)

Developer Guide

The Developer Guide provides resources for developers who want to perform advanced functions or build applications that integrate with Cisco Crosswork Situation Manager.

If you want to build a new integration or create a custom reporting dashboard, this guide outlines how you can expose API endpoints to invoke various actions and functionality.

You can use the following APIs and modules:

- Graze API: Main API that you can use to create, retrieve, and update data in Cisco Crosswork Situation Manager.
- Stats API: API that you can use to retrieve statistics from Cisco Crosswork Situation Manager for reporting and dashboards.
- Topologies API: API that allows you to create, modify, retrieve and delete topologies and their nodes and links.
- Integrations API: Acts as an integration point for external services and exposes selected Cisco Crosswork Situation Manager functionality to authorized external clients.
- Moobot modules: Allows you to create bots to perform automated tasks and expose functions in different Moolets.
- Programmatic LAM: A custom polling LAM that you can use to accept API calls and parse the responses into Cisco Crosswork Situation Manager events.

You can also use these APIs to perform tasks such as Situation enrichment. See Enrichment for more information.

Graze API

You can use the Graze API to perform actions including the following:

- Assign and de-assign alerts.
- Create and close Situations.
- Add processes and services.
- Create and update SAML realms.
- Create and delete maintenance windows.

The Graze API is useful if you want to perform repeated actions. For example, sending repeated cURL commands using a script. For example, you can create a ticketing integration to enable bi-directional communication between Cisco Crosswork Situation Manager and a ticketing system. See Integrations for available ticketing integrations in Cisco Crosswork Situation Manager.Integrations

An integration can raise and close Cisco Crosswork Situation Manager alerts in line with ticketing events, assign users and replicate comments. For all available endpoints see Graze API.

Stats API

You can use the Stats API to retrieve statistics about:

1. Your Cisco Crosswork Situation Manager system's performance.
2. Teams' performance.
3. Individual users' performance.

You can use this statistical data to populate dashboards. For example, you can use call **`getNewEventsPerSituationsStats`** to see the noise reduction from events to Situations for your Cisco Crosswork Situation Manager system.

These endpoints have been designed and optimized for Grafana. See Stats API for all available endpoints. See the Grafana Setup Tutorial for more installation and configuration instructions.

Topologies API

You can use the Topologies API to create, configure and retrieve information about:

- Topologies
- Nodes
- Links

See Topologies API for more information.

Integrations API

You can use the Integrations API to create, configure and retrieve information about:

- Brokers
- Integrations
- Workflows

See Integrations API for more information.

Moobot modules

You can create and configure Moobot modules to perform automated tasks and expose functions including:

- Access external databases.
- Access an external RESTful API via HTTP.
- Read configuration files within LAMbots and Moobots.
- Build a key value dictionary shared across Moobots.
- Query and manipulate entities in the Cisco Crosswork Situation Manager databases. See MoogDb V2 for all available methods.
- Send an email in response to events occurring in Cisco Crosswork Situation Manager.

For more information see Moobot Modules.

Programmatic LAM

The Programmatic LAM is a custom polling LAM. It is an advanced version of the REST Client LAM. The REST Client LAM accepts a single API call and parses the responses it receives into Cisco Crosswork Situation Manager events. The Programmatic LAM can accept multiple calls but you must define the processing yourself in the LAMbot using JavaScript.

For more information see Programmatic LAM.

Alert Rules Engine

The Alert Rules Engine uses business logic to process alerts based on certain conditions.

Note

Cisco recommends using Workflow Engine to enable custom logic and data processing for events, alerts and Situations. Consider carefully if you can implement your logic with the Workflow Engine before you implement and configure the Alert Rules Engine.Workflow Engine

The conditions that the Alert Rules Engine works with generally involve a time-based analysis so that it can process an event in the context of events that happen later. You can define rules in the Alert Rules Engine to hold alerts for a period of time, identify missing alerts or change the state of alerts. For example, common uses of the Alert Rules Engine include:

- Link Up-Link Down: Delays an alert to see if a link recovers.
- Heartbeat Monitor: Detects any missing network health signals.
- Closing Events: Closes events of a particular type or severity.
- Merging: Merges the state of two distinct alerts.
- Configure Alert Rules Engine

Edit the configuration file at **`$MOOGSOFT_HOME/config/moolets/alert_rules_engine.conf`**.

Refer to Alert Rules Engine Reference to see all available properties.

Example Configuration

The following example demonstrates a simple Alert Rules Engine configuration:

```
{
    name               : "AlertRulesEngine",
    classname          : "CAlertRulesEngine",
    run_on_startup     : false,
    metric_path_moolet : true,
    moobot             : "AlertRulesEngine.js",
    process_output_of  : "MaintenanceWindowManager"
}
```

Define Action States and Transitions

The Alert Rules Engine uses Action States and transitions and their properties, to process alerts through business logic defined in the **`AlertRulesEngine.js`** Moobot. After you have configured the Alert Rules Engine, set up Action States and transitions in the Cisco Crosswork Situation Manager UI under Settings > Automation:

- Action States: Determine the length of time Cisco Crosswork Situation Manager retains alerts before forwarding them to a Sigaliser or closing them.
- Transitions: Defines the set of conditions an alert must meet before it moves from one state to another in the Alert Rules Engine. Higher priority transitions take precedence over those with lower priorities.

See Action States and Transitions for further information on how to define them and the properties available.

The initial state for all alerts is the 'Ground' state. After an alert enters 'Ground' state, the Alert Rules Engine transitions it to another state or forwards it to a Sigaliser. If the Action State has a 'Remember Alerts For' set to a positive number, the Alert Rules Engine retains an alert in that state for this period of time.

If you enable 'Cascade on Expiry' and nothing happens to an alert within that period, the Alert Rules Engine returns it to 'Ground' state before forwarding it to a Sigaliser. This is because the 'Ground' state has "Forward Alerts" enabled. If an alert does not match any transitions, the Alert Rules Engine does not return it to 'Ground' state and it is closed.

Note

Action States are not enabled until you have defined a transition.

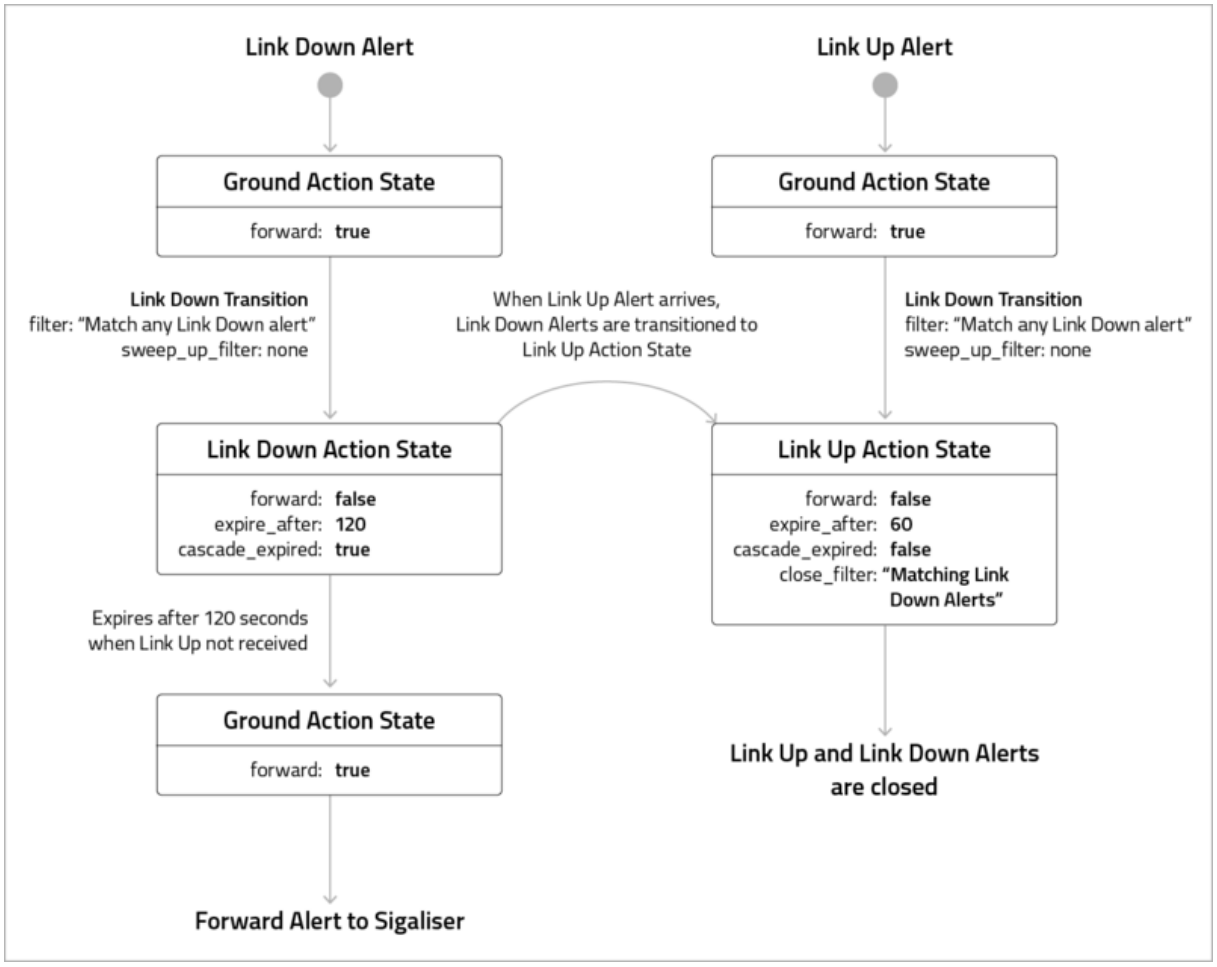
Alert Rules Engine Examples

The Alert Rules Engine can be set up to process Link Up-Link Down events. It can also be set up to act as a Heartbeat Monitor.

Link Up-Link Down Example

This example demonstrates how to configure the Alert Rules Engine so that when a Link Down alert arrives at Moogfarmd, the Alert Rules Engine holds it for a period of time to provide an opportunity for the Link Up alert to arrive. If nothing arrives, the Alert Rules Engine forwards it to a Sigaliser.

If the Link Up alert arrives, the Alert Rules Engine closes and discards both alerts without sending anything to the Sigaliser. This ensures that neither the Link Down nor the Link Up alert appear in Situations.



To try out this example, set up the following:

- Create three Action States: 'Ground' (default), 'Link Up' and 'Link Down'.
- Create two transitions: 'Link Down Transition' and 'Link Up Transition'.

In this scenario, if a 'Link Down' alert arrives at the Alert Rules Engine and no 'Link Up' alert arrives within 120 seconds, the 'Link Down' alert returns 'Ground State' and the Alert Rules Engine passes it to a Sigaliser.

Heartbeat Monitor

You can configure the Alert Rules Engine Moolet in Cisco Crosswork Situation Manager to detect missing heartbeat events from monitoring tools such as CA Spectrum and Microsoft SCOM. Both of these tools send regular heartbeats to indicate normal operation.

After you configure the Alert Rules Engine, Cisco Crosswork Situation Manager creates a Situation when an event source does not send a heartbeat after a given time period. The Alert Rules Engine holds each heartbeat alert for a period of time, subsequent alerts from the same heartbeat source reset the timer. If the timer expires, a heartbeat has been missed and the alert is forwarded to a Sigaliser (clustering algorithm).

Before You Begin

Before you set up the heartbeat monitor in Alert Rules Engine, ensure you have met the following requirements:

- You have an understanding of Alert Rules Engine, Action States and transitions. See the Alert Rules Engine Moolet, Action States and Transitions for further details.
- You can identify heartbeat alerts in the integration by description, class or another configurable field. These must be specific, regular events that arrive at consistent intervals to indicate normal operation. If these are not available, the Heartbeat Monitor will not work.
- You have edited the alerts so they contain the same attribute, via the integration source or through enrichment. In the example below, '**type**' is 'heartbeat' in the Alert Rules Engine trigger filter and '**class**' is 'heartbeat' in the Cookbook Recipe trigger filter.

Create a Heartbeat Monitor

To create a heartbeat monitor in Alert Rules Engine, follow these steps:

- Edit **$MOOGSOFT_HOME/bots/moobots/AlertRulesEngine.js** and add the heartBeatSeverity exit action. This function changes the alert severity to critical and ensures alerts that are closed are not forwarded to the Cookbook. See Status ID Reference for a list of status IDs.

```
function heartBeatSeverity(alert,associated) {
  var currentAlert = moogdb.getAlert(alert.value("alert_id"));
  if ( currentAlert && currentAlert.value("state") !== 9 ) {
    alert.set("severity",5);
    var alertDescr = currentAlert.value("description");
    // Update the description to "MISSED", a successful heartbeat will reset this.
    if ( !/^MISSED/i.test(alertDecr) ) {
      alert.set("description", "MISSED: " + alertDescr)
    }
    moogdb.updateAlert(alert);
    currentAlert.forward("HeartbeatCookBook");
  }
}
```

- Navigate to Settings > Action States in the Cisco Crosswork Situation Manager UI.
- Create a new Action State called "Heartbeat" as follows:

Setting Name	Input	Value
Name	String	Heartbeat
Remember alerts for	Integer (seconds)	30 *
Cascade on expiry	Boolean	True
Exit Action	String	heartBeatSeverity

Warning:

The **Remember alerts for** setting is the timer. Set this to two or three times your heartbeart interval time.

- Go to Settings > Transitions in the Cisco Crosswork Situation Manager UI. Set up a transition to move your heartbeat alerts to the 'Heartbeat' State. Configure the settings as follows:

Setting Name	Value

Name	Heartbeat
Priority	10
Active	True
Trigger Filter	(type = "heartbeat") AND ((((agent = "SPECTRUM") OR (manager= "SCOM")) OR (agent = "MONITOR1")) OR (agent = "MONITOR2"))
Start State	Ground
End State	Heartbeat

Edit the 'Trigger Filter' to meet your requirements. In this example, the transition is triggered by alerts with the type of 'heartbeat' and that come from either 'SPECTRUM' or 'SCOM' or 'MONITOR1' or 'MONITOR2':

- Ensure Alert Rules Engine is enabled. To do this, edit the **`$MOOGSOFT_HOME/config/moolets/alert_rules_engine.conf`** file and set **`run_on_startup`** to true.

- Create a **`heartbeat.conf`** configuration file in **`$MOOGSOFT_HOME/config/moolets`** to add a Heartbeat Cookbook for heartbeat alerts. This only works with these alerts:
```
# Moolet
name:"HeartbeatCookBook",
classname:"CCookbook",
run_on_startup:true,
```

```
    metric_path_moolet : true,
    moobot:"Cookbook.js",
    process_output_of:"[]",
    # Algorithm
    membership_limit:5,
    scale_by_severity:false,
    entropy_threshold:0.0,
    single_recipe_matching:false,
    recipes:[
        # Any heartbeat class for the same agent.
        {
            chef:"CValueRecipe",
            name:"ScomHeartbeatErrors",
            description:"SCOM Heartbeat: Missing heartbeat",
            recipe_alert_threshold:0,
            exclusion:"state = 9",
            trigger:"class = 'heartbeat' AND agent = 'SCOM'",
            rate:0,
            # Given in events per minute
            min_sample_size:5,
            max_sample_size:10,
            matcher:{
                components:[
                    {
                        name:"agent",
                        similarity:1.0
                    }
                ]
            }
        },
        {
            chef:"CValueRecipe",
            name:"ScomHeartbeatChange",
            description:"SCOM Heartbeat: Cluster host change",
            recipe_alert_threshold:0,
            exclusion:"state = 9",
            trigger:"class = 'heartbeatRoleChange' AND agent = 'SCOM'",
            rate:0,
            # Given in events per minute
            min_sample_size:5,
            max_sample_size:10,
            matcher:{
                components:[
                    {
                        name:"agent",
                        similarity:1.0
                    }
                ]
            }
        }
    ],
    cook_for:20000
}
```

- Save **heartbeat.conf**.
- Edit the Moogfarmd configuration file **$MOOGSOFT_HOME/config/moog_farmd.conf** to add a new merge group that references the HeartBeatCookbook Moolet. Configure this merge group to have an **alert_threshold** of 1 to allow a single alert to create a Situation (by default, a minimum of 2 alerts are required to create a Situation):**merge_groups:**

```
        [
```

```
            {
                name: "Heartbeat",
                moolets: ["HeartbeatCookBook"],
                alert_threshold : 1,
                sig_similarity_limit : 1
            }
        ],
```

- Include the Moolet configuration by adding the following in **$MOOGSOFT_HOME/config/moog_farmd.conf**:**{**
  include : "heartbeat.conf"
  },
- Save the changes to **moog_farmd.conf**.|
- Restart Moogfarmd:**service moogfarmd restart**

After the heartbeat monitor configuration is complete, heartbeat alerts should start to arrive in Cisco Crosswork Situation Manager.

Heartbeat Monitor Process

The process flow for a heartbeat alert is as follows:

- Heartbeat alert arrives at the Alert Rules Engine.
- The alert is transitioned from 'Ground' to 'Heartbeat' action state and starts the timer.
- The alert sits in the 'Heartbeat' state waiting for the timer to expire.
- Any subsequent heartbeat alert resets the timer.
- If the timer expires the exit action changes the alert severity to '5' (critical) and cascades it to 'Ground' state.
- Any subsequent heartbeat updates the severity to '0' (clear) and restarts the timer.
- You could also add an entry action to close any missed heartbeat situations the event is part of.

This example also updates the alerts with the times of the missing heartbeats for an easy audit trail.

Alert Rules Engine Reference

This is a reference for the Alert Rules EngineMoolet.

Cisco recommends you do not change any properties that are not in this reference guide.

You can change the behavior of the Alert Rules Engine by editing the configuration properties in the **$MOOGSOFT_HOME/config/moolets/alert_rules_engine.conf** configuration file. It contains the following properties:

name

Name of the Alert Rules Engine Moolet. Do not change.

Type: String

Required: Yes

Default: **"AlertRulesEngine"**

classname

Moolet class name. Do not change.

Type: String

Required: Yes

Default: **"CAlertRulesEngine**"

run_on_startup

Determines whether the Alert Rules Engine runs when Cisco Crosswork Situation Manager starts. By default, it is set to **false**, so it does not start when Moogfarmd starts. You can change this property to **true** so that, when Moogfarmd starts, it automatically creates an instance of the Alert Rules Engine.

Type: Boolean

Required: Yes

Default: **false**

metric_path_moolet

Determines whether or not Cisco Crosswork Situation Manager includes the Alert Rules Engine in the Event Processing metric for Self Monitoring.Self Monitoring

Type: Boolean

Required: Yes

Default: **true**

moobot

Specifies a JavaScript file found in **$MOOGSOFT_HOME/moobots**, which defines the Alert Rules Engine Moobot. The default, **AlertRulesEngine.js**, provides the standard modules. You can customize it to meet your needs.

Type: String

Required: Yes

Default: **"AlertRulesEngine.js"**

mooms_event_handler

Determines whether or not the Alert Rules Engine listens for messages on the message bus. If set to **true**, the Alert Rules Engine processes messages on the Alerts topic on the message bus. This property should not be included in the configuration file, or should be commented out, if the **process_output_of** property is defined.

Type: Boolean

Required: No

Default: **false**

process_output_of

Defines the input source for the Alert Rules Engine. This determines the Alert Rules Engine's place in the alert processing workflow. If this property is defined, the **`mooms_event_handler`** property should be omitted or commented out in the configuration file.

Type: List

Required: No

One of: **`AlertBuilder`**, **`MaintenanceWindowManager`**, **`Enricher`**

Default: **`"MaintenanceWindowManager"`**

Transitions

Transitions are a user-configurable set of conditions that move an alert from one state to another within the Alert Rules Engine (ARE). Transitions result in the following:

- Alerts move from one Moolet to the next Moolet in the chain.
- Alerts pass to a clustering algorithm which clusters them into Situations.

Field	Input	Description
Name	String	Name of the transition. This can be up to 64 characters. Mandatory.
Description	String	Description of the transition.
Priority	Integer	Determines the priority of the transition if there are multiple transitions. The higher the value, the higher the priority.
Active	Boolean	Sets the transition to active.
First Match Only	Boolean	Transition only occurs once if an alert meets the trigger conditions.
Trigger Filter	Filter	Filter that triggers the transition if an alert meets the defined trigger filter parameters. Mandatory.
Inclusion Filter	Filter	Filter that passes additional alerts to the end state if they arrive after the initial trigger and meet the defined inclusion filter parameters.
Start State	-	Determines the action state of the alerts in the inclusion filter. The start state and end state must be different. Mandatory.
End State	-	Determines the action state of the alerts if they match the inclusion or trigger filters. The start state and end state must be different. Mandatory.

To create and configure different transitions go to System Settings.

Create a New Transition

Click + to create a new transition and edit the fields to meet your requirements:

When you have configured the transition, click Save to continue.

Delete a Transition

To delete a transition from the list of available transitions:

- Select the transition to delete.
- Click - to delete the transition.
- Click Yes to confirm the deletion.

Action States

Action States are the different states which alerts are placed into as they pass from the Alert Builder into the Alert Rules Engine. See the Alert Rules Engine for a standard link up-link down example that uses Action States.

The different states define how long the alerts are retained in a certain state and whether they are forwarded to any Moolets or Sigalisers. The default or base state is called 'Ground'. It is required for the system to function correctly and cannot be deleted. This is the state that alerts have when they enter the Alert Rules Engine.

Create an Action State

Click + to create a new Action State. The available fields are as follows:

Field	Input	Description
Name	String (Mandatory)	Name of the new Action State (up to a maximum of 64 characters).
Description	String	Description of the new Action State.
Remember Alerts For	Integer	Time in seconds that the system remembers the alerts in this state for. Any number less than 0 (<0) means do not remember it, so the state never retains a memory of the alert. 'Ground' has -1 because you do not want to accumulate a memory of every alert in the system. By default, you want the alert to pass to a Sigaliser. The purpose of the state engine is to spot specific alerts and do different things with them.
Cascade on Expiry	Boolean	Specifies what to do if you have set a time to remember alerts for. For example, the alert goes into the state and then after the set time of 30 seconds it is taken out of the state whether you dispose of it manually or return it back to its original state.
Forward Alerts	Boolean	If enabled, the alerts that enter this state are forwarded to the chain Moolet.
Close Filter	Filter	Defines which alerts are closed when they enter the state.
Entry Action	String	Moobot function that is called when an alert enters the state.
Exit Action	String	Moobot function that is called when an alert exits the state.

Click Save Changes to continue. Alternatively, click Revert Changes to undo any changes. The new Action State appears on the list to the left.

Delete an Action State

Select the Action State you want to delete from the list on the left.

Click - to delete the Action State. The pop-up confirmation window appears. Click Yes to confirm the deletion.

Clustering Algorithm Guide

Sigalisers are the clustering algorithms in Cisco Crosswork Situation Manager that group alerts based on factors such as time, language, similarity and proximity.

The clustering algorithms available include:

- Cookbook
- Tempus

You can configure and run multiple different clustering algorithms on the same instance of Cisco Crosswork Situation Manager. The algorithms you choose depend on your specific use cases and the type of Situations you want your operators to receive.

You can also apply entropy and Vertex Entropy calculations to add another degree of filtering to the alerts you want to correlate. For example, you can use an entropy threshold if you want to exclude alerts with low operational value or include alerts with high operational value. See Vertex Entropy and Entropy for more details.

Cookbook

Cookbook is a clustering algorithm that creates clusters defined by the relationships between alerts and their attributes. See Cookbook for more information.

Type: Attribute-based clustering.

Use cases: You can use Cookbook if you want more control in how you correlate alerts based on patterns in the text similarity. Example use cases include:

- Grouping alerts with a similar description and from the same application or service.
- Grouping alerts from the same host or location.
- Topology-based correlation using Vertex Entropy.

Benefits: Cookbook offers the following advantages:

- Very customizable and configurable using Recipes.
- Able to create Situations when an alert exceeds a defined rate of occurrence.
- Can include and exclude alerts that meet specific criteria such as Vertex Entropy.
- Able to partition alerts into Situations using textual similarity-based comparison.
- Possible to base alert clustering on topological relationships.

Configuration: To configure Cookbook Recipes and Cookbook via the Cisco Crosswork Situation Manager UI, see Configure a Cookbook Recipe and Configure a Cookbook. You can also configure Cookbook and its Recipes via the Graze API.

Tempus

Tempus is a time-based algorithm that clusters alerts into Situations based on the similarity of their timestamps. See Time-Based Clustering with Tempus for more information.

Type: Time-based clustering.

Use cases: You want to match alerts based on patterns in their timestamps or on a timeline. Use Tempus if you want your alerts to be clustered in real-time. The logic behind Tempus is that a triggering event causes additional subsequent failures within a short timeframe. Works well in scenarios where there is a causal chain such as:

- Cascading failures
- Performance failures
- Brownouts.

Benefits: Tempus offers the following advantages:

- No enrichment required. See Enrichment
- Good for availability alerts.
- Good for performance alerts.

Configuration: To configure Tempus via the Cisco Crosswork Situation Manager UI, see Configure Tempus. You can also configure Tempus via the Graze API.

Situation Manager Labeler

You can use the Situation Manager Labeler to set Situation descriptions and fields dynamically, based on the alert data in each Situation. For example, suppose you are defining a correlation based on the **`custom_info.services`** alert field. To generate descriptions for the resulting Situations, you can specify a label string in the description field such as:

```
$$COUNT(custom_info.services) services affected including
$$CITED(custom_info.services,3)
```

Given this string, the resulting descriptions include the three most-cited services and the number of times each service is cited by a member alert:

```
5 services affected including cust-login(7), verify-login(6), update-login-
info(4), ...
```

Note

The Situation Manager Labeler is installed by default with Cisco Crosswork Situation Manager v7.3 and higher. For previous releases, contact Cisco Customer Support to obtain installers and instructions.

Usage

Given a macro operation and an alert data field, the operation iterates through the relevant values in the Situation alerts and returns a string derived from these values.

The usage for fields with single values (prefix is one $): **$macro(alert-field, max-alerts-to-include)**

The usage for fields with lists (prefix is two $'s): **$$macro(alert-field, max-alerts-to-include)**

The **max-alerts-to-include** field is optional. This value limits the number of alert values to include in the description.

Consider the following example. You want to create a label with a count of all the affected services (**custom_info.services**) cited in all alerts. A Situation has two alerts:

- Alert 1 - **custom_info.services = [a, b, c];**
- Alert 2 - **custom_info.services = [d, e, f];**

$COUNT treats the fields as individual values and returns a count of 2.

$$COUNT treats the fields as lists of individual values and returns a count of 6.

Update Situation descriptions

You can use the following macros to generate Situation descriptions. These macros are supported for single values (**$macro**) and lists (**$$macro**):

- `COUNT(alert-field)` -– Return the count of alert-field citations, including duplicates.
- `UCOUNT(alert-field)` -– Return the count of unique alert-field citations, excluding duplicates.
- `CRITICAL(alert-field)` -– Return the string **CRITICAL :** if any alerts have a severity of critical, or 5. This macro is only useful for the **severity** field.
- `UNIQ(alert-field)` -– Return a list of all cited **alert-field** values.
- `TOP(alert-field)` -– Return the **alert-field** value cited by the most alerts in the Situation.
- `CITED(alert-field)` -– Return a list of the unique **alert-field** values cited by alerts in the Situation along with the number of times they are cited -- for example, **source1 (10)**, **source5 (7)**, **source3 (4)**.
- `CITEDLIST(alert-field)` -– Same as **$CITED** but returns a string instead of a JSON list.
- `BOOLEAN(alert-field)` -– Return false if all values are "falsy:" 0, null, undefined, "", and so on.
- `TOLIST(alert-field)` -– Creates a comma-separated string from the elements of **alert-field**.

Note

UI list-based filtering is now native, so **$TOLIST()** should no longer be required.

Numeric fields only

The following macros are supported for numeric fields only, such as **time**, **severity**, or **event-count**.

- `MIN(alert-field)` -– Return the minimum cited value of **alert-field**.

- `MAX(alert-field)` -– Return the maximum cited value of **`alert-field`**.
- `AVE(alert-field)` -– Return the average of all cited values of a**`alert-field`**.
- `SUM(alert-field)` -– Return the average of all cited values of **`alert-field`**.
- `NUM(alert-field)` -– Return the set of **`alert-field`** values sorted numerically from low to high, including duplicates.
- `UNUM(alert-field)` -– Return the set of unique **`alert-field`** values sorted numerically from low to high, excluding duplicates.

Text fields only

The following macros are supported for text fields only, such as **`service`**, **`source`**, or **`description`**.

- `ALPHA(alert-field)` -– Return the set of alert-field values sorted alphabetically, including duplicates.
- `UALPHA(alert-field)` -– Return the set of unique alert-field values sorted alphabetically, excluding duplicates.

List values only

The following macros are supported for array values only.

- `$$INTERSECT(alert-field)` -– Return the list of intersections -- that is, alert-field values cited by multiple alerts. This macro parses the alert-field array values and returns a list of the items with multiple citations.For example, support a Situation has two alerts. The service field of alert 1 is **`[a, b, c]`**. The service field of alert 2 is **`[b, c, d]`**. **`$$INTERSECT(service)`** would return the list **`[b, c]`**.
- `$$NINTERSECT(alert-field)` -– Return the number of intersections. Given the previous example, **`$$NINTERSECT(service)`** would return the number 2.
- `$$CINTERSECT(alert-field)` -– Return the list of common intersections -- that is, values cited by all alerts in the Situation. This macro is useful for identifying a possible root cause that caused all the alerts to get correlated together.

Limiting the number of alerts to consider

By default, each macro considers all alerts in a Situation up to a maximum of 200. You might want to specify a lower threshold to ensure that labeling does not become a bottleneck in systems with large or frequently-updated Situations. To lower the threshold, append the **`$FETCH`** modifier at the start of the Labeler string:

```
$FETCH(max-alerts-to-consider)Labeler-string
```

For example, the following macro considers the first alert in each Situation based on alert ID:

```
$FETCH(1) Application Situation for: $UNIQ(custom_info.application) at
DataCentre $UNIQ(custom_info.location)
```

You should specify the maximum number of alerts needed to ensure an accurate description. If you are correlating based on a specific field such that all alerts have the same value for that field, you only need to fetch 1 alert.

Warning

Do not specify a fetch value higher than 20.

Update other Situation fields

You can use the following macros to update columns and fields in a Situation with values contained in its member alerts.

- `$$SERVICES(alert-field)`-– Update the Services Impacted column in the Situation with all unique alert-field values cited in the member alerts.
- `$$ISERVICES(alert-field)` -– Update the Services Impacted column in the Situation with all unique alert-field values cited in 2 or more member alerts.
- `$$PROCESSES(alert-field)`-– Update the Processes Impacted column in the Situation with all unique alert-field values cited in the member alerts.

You can also use the **`$MAP[ ]`** macro to update a **`custom_info`** field in the Situation with data from the member alerts. The usage is as follows:

`$MAP[ $MACRO(source alert field, destination custom_info field) ]`

You can include multiple macros in the same MAP macro, as shown in the following example:

```
$MAP[ $UNIQ(source, hosts) $UCOUNT(source, num_hosts) ]
```

Example

For instructions on how to use the Situation Manager Labeler to automatically create services based on custom_info data, see Create Situation using Situation Labeler.

Field Behavior in Merged Situations

When Cisco Crosswork Situation Manager merges two or more Situations, it updates the fields of the Situations as follows:

Field	Old Situations	New Situation
Category	Superseded.	Created.
Created At	No change.	Time of merge.
Description	No change.	Merge of Situations [X, Y, Z] where X, Y, and Z represent the Situation IDs of the superseded Situations.
First Event Time	No change.	The First Event Time for the combined Situations.
ID	No change.	The next sequential Situation ID.
Last Change	No change.	The time that the merge took place.
Last Event Time	No change.	The value of the Situation in first position in the merge list.
Owned By	No change.	Default (none).
Participants	No change.	Default (none).

Process Impacted	No change.	Combined values.
Queue	No change.	The queue of the Situation in first position in the merge list.
Rating	No change.	Default (none).
Scope	No change.	Combined values.
Scope Trend	No change.	Combined values.
Services Impacted	No change.	Combined values.
Sev Trend	No change.	Combined values.
Severity	No change.	The highest severity of the merged Situations.
Status	Dormant.	Opened.
Story	Adopts ID of new Situation.	The Story ID is the same as the Situation ID of the new Situation.
Teams	No change.	All Teams monitoring the merged Situations.
Total Alerts	No change.	The sum of the Alerts of all merged Situations.
User Comments	No change.	Default (none).

Graze API

The Graze API acts as an integration point for external services and exposes selected Cisco Crosswork Situation Manager functionality to authorized external clients.

Use caution when employing the Graze API. Excessive requests can impact overall system performance, especially **`getSituationIds`** and **`getAlertIds`**.

Contact Cisco Support if you experience difficulties or need further guidance.

Endpoints

See Graze API EndPoint Reference for details of all the Graze API endpoints.

Before you begin

Cisco Crosswork Situation Manager implements the Graze API as a set of servlets running in the Cisco Crosswork Situation Manager Apache Tomcat instance. This instance handles external Graze requests, making the UI servlet calls directly via cross-contexts.

Configure Apache Tomcat

You must configure Apache Tomcat to allow cross-context calls to be made by adding the following to the **`context.xml`** file in the Apache Tomcat **`$APPSERVER_HOME/conf`** directory:

```
<Context crossContext="true">
```

API definition

All Graze requests use the following URL format, where **<server>** is the hostname of the machine running the UI :

```
https://<server>/graze/v1/<request_type>
```

For example:

```
https://localhost/graze/v1/authenticate
```

Authentication

All Graze API requests, other than **authenticate**, require a basic authentication header or a valid **auth_token**. You must make a valid authenticate request before using any Graze API request without a basic authentication header.

If you make regular Graze requests within a one hour timeframe, you are considered active and your session does not expire. Inactive sessions are logged out after one hour, and you must make a new **authenticate** request to get a new valid **auth_token**.

Authentication troubleshooting

If an error occurs during Graze login authentication, Cisco Crosswork Situation Manager returns the following output:

```
{"message":"User is not authenticated","statusCode":3001}
```

As a security precaution, no more specific information is returned. This prevents information being provided to potential attackers about which part of the authentication failed (for example 'Password incorrect').

Entries in the log file **catalina.out**, at **WARN** level, provide more information on authentication errors:

- For example, the user is not assigned the Grazer role:

```
User [john] does not have graze permission
```

- For example, no user of that name exists:

```
User [NotAUser] account unknown in database
```

- For example, incorrect password:

```
Password incorrect for user [graze]
```

POST parameters

You can send POST parameters as **form-urlencoded** or as **application/json** parameters.

form-urlencoded

To send POST parameters as **form-urlencoded** parameters, set the content type to **application/x-www-form-urlencoded**. If the character set is not set, UTF-8 is assumed.

Example cURL command:

```
"https://localhost/graze/v1/resolveSituation?auth_token=b40244fd79aa46fba76c60c5
6d538c49&sitn_id=10" --insecure -X POST -v
```

application/json

To supply POST parameters as JSON within the body of the request, set the content type to **`application/json`**. If the character set is not set, UTF-8 is assumed.

Example cURL command:

```
"https://localhost/graze/v1/resolveSituation" -H "Content-Type:
application/json; charset=UTF-8" --insecure -X POST -v --data '{"auth_token" :
"b40244fd79aa46fba76c60c56d538c49","sitn_id" : 10}'
```

Graze API EndPoint Reference

This is a reference list for the Graze API endpoints. Follow the links to see the details of each endpoint.

All Graze API requests, other than authenticate, require a basic authentication header or a valid **`auth_token`**. You must make a valid authenticate request before using any Graze API request without a basic authentication header. See Authentication for more information.

Alerts

The following Graze API endpoints relate to alerts:

- addAlertCustomInfo: Adds and merges custom information for an alert.
- addAlertToSituation: Adds an alert to a Situation.
- assignAlert: Assigns a user as the owner of an alert.
- assignAndAcknowledgeAlert: Assigns and acknowledges a user as the owner of an alert.
- closeAlert: Closes one or more alerts.
- deassignAlert: Deassigns the current owner from an alert.
- getAlertActions: Returns the actions for one or more alerts.
- getAlertDetails: Returns details, such as the description or severity, of an alert.
- getAlertIds: Returns the total number of alerts, and a list of the alert IDs, for an alert filter and a limit.
- removeAlertFromSituation: Removes an alert from a Situation.
- resolveAlerts: Resolves a list of alerts.
- setAlertAcknowledgeState: Acknowledges or unacknowledges the owner of an alert.
- setAlertSeverity: Sets the severity level of an alert.
- updateClosedAlert: Updates the description and custom info of a closed alert during the grace period.

Algorithms

Use the following Graze API endpoints to configure Cookbooks and Recipes:

- addBotRecipe: Creates a new Cookbook Bot Recipe.

- addCookbook: Creates a new Cookbook.
- addValueRecipe: Creates a new Cookbook Recipe using Value Recipe or Value Recipe v2 recipe types.
- deleteCookbook: Deletes an existing Cookbook.
- deleteRecipe: Deletes an existing Cookbook Recipe.
- getCookbooks: Returns all the Cookbooks in Cisco Crosswork Situation Manager.
- getRecipes: Returns all the Recipes in Cisco Crosswork Situation Manager.
- updateBotRecipe: Updates a Cookbook Bot Recipe.
- updateCookbook: Updates a Cookbook.
- updateValueRecipe: Updates a Cookbook Recipe that uses either a Value Recipe or a Value Recipe v2 recipe type.

Use the following Graze API endpoints to configure Tempus:

- addTempus: Adds a new Tempus Moolet.
- deleteTempus: Deletes an existing Tempus Moolet.
- getTempus: Returns the details of all Tempus Moolets in Cisco Crosswork Situation Manager.
- updateTempus: Updates an existing Tempus Moolet.

Use the following Graze API endpoints to update the default merge group:

- getDefaultMergeGroup: Returns details of the default merge group in Cisco Crosswork Situation Manager.
- updateDefaultMergeGroup: Updates the default merge group in Cisco Crosswork Situation Manager.

Use the following Graze API endpoints to configure custom merge groups:

- addMergeGroup: Adds a new custom merge group.
- deleteMergeGroup: Deletes an existing custom merge group.
- getMergeGroups: Returns details of all the custom merge groups in Cisco Crosswork Situation Manager.
- updateMergeGroup: Updates a custom merge group.

Dashboards and reporting

See the Stats API for information on Graze API endpoints that provide statistics related to dashboards or reporting.

Entropy thresholds and Events Analyser configuration

Use the following Graze API endpoints to set entropy thresholds:

- getGlobalEntropyThresholds: Returns the global default entropy threshold and all manager-specific entropy thresholds that have been set.
- setGlobalEntropyThreshold: Sets the global default entropy threshold or a manager-specific entropy threshold.
- Use the following Graze API endpoints to configure the Alert Analyzer:
- getEventsAnalyserConfig: Returns the list of priority words or stop words used by the Alert Analyzer.
- updateEventsAnalyserConfig: Updates the Alert Analyzer configuration.

If you want to set up priority word or stop word lists in the Alert Analyzer, use the following Graze API endpoints:

- addEventsAnalyserWord: Adds a single word to a list of priority words or stop words in the Alert Analyzer configuration.
- getEventsAnalyserWords: Returns the list of priority words or stop words used by the Alert Analyzer.
- removeEventsAnalyserWord: Removes a single word from the list of priority words or stop words in the Alert Analyzer configuration.
- updateEventsAnalyserWords: Updates an existing list of priority words or stop words in the Alert Analyzer configuration.

If you want to set up partitions in the Alert Analyzer, use the following Graze API endpoints:

- getEventsAnalyserPartitionOverrides: Returns the partition override details in the Alert Analyzer configuration.
- removeEventsAnalyserPartitionOverrides: Removes all the partition overrides from the Alert Analyzer configuration.
- updateEventsAnalyserPartitionOverrides: Updates the partition overrides in the Alert Analyzer configuration.

Processes and maintenance

The following Graze API endpoints relate to Cisco Crosswork Situation Manager processes and scheduled maintenance:

- addProcess: Adds a new process to the database.
- addService: Adds a new external service to the database.
- createMaintenanceWindow: Creates a maintenance window that filters alerts caused by a known period of maintenance.
- deleteMaintenanceWindow: Deletes a single maintenance window.
- deleteMaintenanceWindows: Deletes maintenance windows that match a filter.
- Match List Items in Recipes: Returns maintenance windows that match a filter.
- getIntegrationConfig: Exports the configuration and mapping needed for an integration in JSON format.

- getMaintenanceWindows: Returns maintenance windows based on the window ID and how many should be fetched.
- getProcesses: Returns a list of the processes in the database.
- getServices: Returns a list of the services in the database.
- getSeverities: Returns a list of possible severities and their severity IDs.
- getStatuses: Returns a list of statuses that can apply to Situations and their IDs.
- getSystemStatus: Returns current system status information for all processes.
- getSystemSummary: Returns a summary of current alerts and Situations in Cisco Crosswork Situation Manager.
- getToolShares: Returns the shared access for a tool.
- shareToolAccess: Shares access to a tool with other users, teams, or roles, or makes it global so that all users can access it.
- updateMaintenanceWindow: Updates an existing maintenance window.

Situations

The following Graze API endpoints relate to Situations:

- addSigCorrelationInfo: Associates the external client with a Situation.
- addSituationCustomInfo: Adds and merges custom information for a Situation.
- addThreadEntry: Adds a new entry to an existing thread in a Situation.
- assignAndAcknowledgeSituation: Assigns and acknowledges the moderator to a Situation.
- assignSituation: Assigns the moderator to a Situation.
- assignTeamsToSituation: Assigns one or more teams to a Situation, or unassigns all teams from a Situation.
- closeSituation: Closes a Situation which is currently open, and optionally closes alerts in the Situation.
- createSituation: Creates a manual Situation. The Situation description is set with the **`description`** parameter.
- createThread: Creates a new thread for a Situation.
- createThreadEntry: Creates a new entry in an existing thread in a Situation. This endpoint has been superseded by addThreadEntry.
- deassignSituation: Deassigns the current moderator from a Situation.
- getActiveSituationIds: Returns the total number of active Situations, and a list of their Situation IDs.
- getPrcLabels: Returns probable root cause (PRC) information for all alerts or specified alerts within a Situation.

- getResolvingThreadEntries: Returns thread entries for a Situation that have been marked as resolving steps.
- getSigCorrelationInfo: Returns all correlation information related to a Situation.
- getSimilarSituationIds: Returns a list of IDs of similar Situations, for a Situation ID and a limit.
- getSimilarSituations: Returns the details of similar Situations for a Situation and a limit.
- getSituationActions: Returns the actions for a list of Situations.
- getSituationAlertIds: Returns the total number of alerts, and a list of the alert IDs for a Situation.
- getSituationDescription: Returns the description for a Situation.
- getSituationDetails: Returns the details of a Situation.
- getSituationHosts: Returns the hosts for a Situation.
- getSituationIds: Returns the total number of Situations, and a list of their Situation IDs, for a filter and a limit.
- getSituationPrimaryTeam: Returns the primary team on a Situation.
- getSituationProcesses: Returns a list of process names for a Situation.
- getSituationServices: Returns a list of external service names for a Situation.
- getSituationSeverityChanges: Returns the changes in severity for a Situation. It returns increases in severity and a change to a severity of 0 (Clear).
- getSituationTopology: Retrieves the node and link details for a Situation and topology.
- getSituationVisualization: Returns information on the origin and cause of a Situation.
- getThreadEntries: Returns thread entries for a Situation.
- getThreadEntry: Returns a thread entry specified using the thread entry ID.
- getTopPrcDetails: Returns the top most likely causal alerts, based on their Probable Root Cause value, for a Situation.
- mergeSituations: Merges multiple Situations.
- rateSituation: Applies a rating to a Situation.
- removeSigCorrelationInfo: Removes all correlation information related to a Situation.
- removeSituationPrimaryTeam: Removes the primary team from a Situation.
- resolveSituation: Resolves a Situation that is currently open.
- setPrcLabels: Sets the probable root cause (PRC) labels for specified alerts within a Situation.
- setResolvingThreadEntry: Sets or clears a thread entry in a Situation as a resolving step.
- setSituationAcknowledgeState: Acknowledges or unacknowledges the moderator who has been assigned to a Situation..
- setSituationDescription: Sets the description for a Situation.

- setSituationPrimaryTeam: Sets one of the teams already assigned to a Situation as the primary team.
- setSituationProcesses: Applies a list of processes to a Situation.
- setSituationServices: Applies a list of external services to a Situation.
- /situation/{situationID}/topologies: Retrieves the topologies related to the alerts in a Situation.
- updateClosedSituation: Updates the description and custom info of a closed Situation during the grace period.

Security realms

The following Graze API endpoints relate to security realms:

- createSecurityRealm: Creates a new security realm from an Identity Provider (IdP) URL.
- getSecurityRealm: Returns a JSON object containing the names and configuration details of active security realms.
- updateSecurityRealm: Updates an existing security realm in the database.

Topologies

See Topologies API Endpoint Reference for a list of endpoints related to topologies.

User management

The following Graze API endpoints relate to the management of users, teams and roles:

- applyNewLicense: Adds a Cisco Crosswork Situation Manager license via Graze.
- createTeam: Creates a new team.
- createUser: Creates a new user.
- deleteTeam: Deletes a single team.
- getAllSessionInfo: Returns session information for all users over a period of time.
- getTeam: Returns a team's details by team ID or name.
- getTeams: Returns the details of all the teams in Cisco Crosswork Situation Manager.
- getTeamsForService: Returns all teams related to the service with the specified ID or name.
- getTeamSituationIds: Returns the total number of Situations that are assigned to a team, and a list of their Situation IDs.
- getUserInfo: Returns information about a user.
- getUserRoles: Returns the user's roles from the database.
- getUsers: Returns a list of all users in the database.
- getUserSessionInfo: Returns session information for a single user over a period of time.

- getUserTeams: Returns the team names and IDs associated with a user ID or username.
- updateTeam: Updates an existing team.
- updateUser: Updates an existing user.

Workflow Engine

The following Graze API endpoints relate to the Workflow Engine:

- createWorkflow: Creates a new workflow in the Workflow Engine.
- deleteWorkflow: Deletes a workflow from the Workflow Engine.
- getWorkflowEngineMoolets: Returns a list of all the workflows in all the Workflow Engine Moolets in Cisco Crosswork Situation Manager.
- getWorkflows: Returns workflows for a specified Workflow Engine Moolet.
- reorderWorkflows: Reorders the sequence of workflows within a Workflow Engine Moolet.
- sendToWorkflow: Sends a Moolet Inform message to a workflow in an Inform Workflow Engine.
- updateWorkflow: Updates an existing workflow in the Workflow Engine.

addAlertCustomInfo

A Graze API POST request that adds and merges custom information for a specified alert.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`addAlertCustomInfo`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`alert_id`	Number	Yes	Alert ID.
`custom_info`	JSON object	Yes	A JSON object containing the custom information.

Response

Endpoint **`addAlertCustomInfo`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes

Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`addAlertCustomInfo`**:

Request example

Example cURL request to add custom info to `"field1"`, `"field2"`, `"field3"`, and `"field4"` in alert ID 9:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/addAlertCustomInfo" -H "Content-Type:
application/json; charset=UTF-8" -d '{"alert_id" : 9, "custom_info" : { "field1"
: "value1" , "field2" : "value2" , "field3" : ["item1","item2","item3"] ,
"field4" : {"field4-1" : "value4-1","field4-2" : "value4-2"} }}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addAlertToSituation

A Graze API POST request that adds a specified alert to a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`addAlertToSituation`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`alert_id`	Number	Yes	Alert ID.
`sitn_id`	Number	Yes	Situation ID.

Response

Endpoint **`addAlertToSituation`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation

Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

This endpoint does not add the alert to the Situation if the alert has been archived to the historic database even if the Situation is still in the active database.

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **addAlertToSituation**:

Request example

Example cURL request to add alert ID 16 to Situation ID 7:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/addAlertToSituation" -H "Content-Type:
application/json; charset=UTF-8" -d '{"alert_id" : 16, "sitn_id" : 7 }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addBotRecipe

A Graze API POST request that creates a new Cookbook Bot Recipe. To create Recipes using the Value Recipe and Value Recipe v2 recipe types, use addValueRecipe. See Recipe Types for more information.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addBotRecipe** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
cookbooks	Array of Strings	No	A list of the Cookbooks that this Recipe belongs to. You can add Cookbooks here or, when you create a Cookbook, you can assign the Recipes to it.
name	String	Yes	Name of the Recipe. Use a unique and descriptive name.
description	String	No	Description of the Recipe. Default is the Recipe **name**.
alert_threshold	Positive Integer	No	Minimum number of alerts required before Cookbook creates a Situation.
trigger	String	No	A filter that determines the alerts that Cookbook considers for Situation creation. Cookbook includes alerts that match the trigger filter. By default Cookbook only includes alerts with a severity of 'Critical'.
exclusion	String	No	A filter that determines the alerts to exclude from Situation creation. Cookbook ignores alerts that match

			the exclusion filter.
seed_alert	String	No	A filter that determines whether to create a Situation from a seed alert. The seed alert must meet both the **trigger**, **exclusion** and **seed_alert** criteria to create a Situation. Cookbook considers subsequent alerts for clustering if they meet the trigger and exclusion filter criteria. Alerts that arrive prior to the seed alert that met the trigger and exclusion filter criteria do not form Situations.
rate	Double	No	Rate, in number of alerts per second. Cookbook clusters alerts if they arrive at a higher rate than is specified here. Cookbook uses **rate** together with **min_sample_size** and **max_sample_size** to determines whether to cluster alerts into Situations. See Cookbook and Recipe Examples. Default is 0 which means that Cookbook does not use the rate to cluster alerts.Cookbook and Recipe Examples
min_sample_size	Positive Integer	No	Number of alerts that must arrive before the Cookbook starts to calculate the alert rate. See Cookbook and Recipe Examplesfor more information. Default is 5. Valid only if **rate** is non-zero.Cookbook and Recipe Examples
max_sample_size	Positive Integer	No	Maximum number of alerts that are considered in the alert rate calculation. When more than this number of alerts have arrived, Cookbook discards the oldest alerts and calculates the alert rate based on the number of alerts in the **max_sample_size**. See Cookbook and Recipe Examplesfor more information. Default is 10. Valid only if **rate** is non-zero.Cookbook and Recipe Examples
cook_for	Positive Integer	No	Minimum time period, in seconds, that the Cookbook Recipe clusters alerts for before it resets and starts a new cluster. See Cookbook and Recipe Examples for more information.Cookbook and Recipe Examples If you set a different **cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for** time inherit the value from the Cookbook.
cook_for_extension	Positive Integer	No	Time period that the Cookbook Recipe can extend clustering alerts for before it resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Cookbook. As Cookbook receives related alerts, it continues to extend the total clustering time until the **max_cook_for** period is reached. Used in conjunction with the **max_cook_for** value, the **cook_for_extension** period helps to ensure that Cookbook continues to cluster alerts together that are related to the same failure. The

			cook_for_extension period only applies to new related alerts; it does not apply to existing alerts that are updated with new events. See Cookbook and Recipe Examples If you set a different **cook_for_extension** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for_extension** time inherit the value from the Cookbook.
max_cook_for	Positive Integer	No	Maximum time period that the Cookbook Recipe clusters alerts for before it resets and starts a new cluster. It works in conjunction with the **cook_for_extension** time to help ensure that Cookbook continues to cluster alerts together that are related to the same failure. This value is ignored unless the **cook_for_extension** time is specified. See Cookbook and Recipe Examples If you set a different **max_cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **max_cook_for** value inherit the value from the Cookbook.
cluster_by	String	No	Determines Cookbook's clustering behavior. Set to an empty string to use the Cookbook **cluster_by** setting. Set to **first_match** so that Cookbook adds alerts to the first cluster over the similarity threshold value. Set to **closest_match** to add alerts to the cluster with the highest similarity greater than the similarity threshold value. This option may be less efficient because Cookbook needs to compare alerts against each cluster in a Recipe. Default is an empty string which means the Recipe uses the Cookbook setting. If you set a different **cluster_by** value for a Recipe, it overrides the Cookbook value. Recipes without a **cluster_by** value inherit the value from the Cookbook.
initialize_function	JSON Function Name	No	Default is **initBuckets**.
member_function	JSON Function Name	No	Default is **checkBucket**.
can_start_cluster	JSON Function Name	No	Default is null.
use_in_recipe	JSON Function Name	No	Name of the function that will determine whether to consider an event for clustering (similar to a trigger filter). Default is null.
similarity	Double	No	Value between 0 and 1. Default is 0.8.

Response

Endpoint **addBotRecipe** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
id	Integer	ID of the new Bot Recipe.

Examples

The following examples demonstrate typical use of endpoint **addBotRecipe**:

Request example

Example cURL request to create a new Bot Recipe "BotRecipe2":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addBotRecipe" -H
"Content-Type: application/json; charset=UTF-8" -d '{"cookbooks" :
["GrazeCookbook1"],"name":"BotRecipe2","alert_threshold":1}'
```

Response example

Successful response providing the ID of the new Bot Recipe that has been created:

```
{
    "id": 4
}
```

addCookbook

A Graze API POST request that creates a new Cookbook.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addCookbook** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the Cookbook. Must be unique.
description	String	No	Description of the Cookbook.
process_output_of	List of Strings	No	Defines the source of the alerts that Cookbook processes. You can specify none, one or more Moolets. Typically Cookbook processes the output of its direct upstream neighbor in the processing chain. Usually this is "Alert Workflows" which are the output from the Alert Workflow Engine.

cluster_by	String	No	Determines Cookbook's clustering behavior. Set to **first_match** so that Cookbook adds alerts to the first cluster over the similarity threshold value. Set to **closest_match** to add alerts to the cluster with the highest similarity greater than the similarity threshold value. This option may be less efficient because Cookbook needs to compare alerts against each cluster in a Recipe. Default is **first_match**. If you set a different **cluster_by** value for a Recipe, it overrides the Cookbook value. Recipes without a **cluster_by** value inherit the value from the Cookbook.
entropy_threshold	Number	No	Minimum entropy value an alert must have in order for Cookbook to consider it for clustering it into a Situation. A value between 0 and 1. Only relevant if **threshold_type** is set to **explicit_value**. If used, Cookbook does not cluster any alerts with an entropy value below the threshold into Situations. Default is 0.0 which means that Cookbook processes all alerts.
threshold_type	String	No	Type of entropy threshold you want Cookbook to use. One of:**global**: Use the global entropy threshold. This is a single entropy threshold that Cookbook applies to all alerts to eliminate noisy alerts with a lower entropy value.**manager**: Use entropy thresholds set up for individual managers. If the manager for an alert has an entropy threshold set, Cookbook uses this value to eliminate noisy alerts with a lower entropy value. If an alert's manager does not have an entropy threshold, Cookbook uses the global entropy threshold to filter out alerts.**explicit_value**: Use the value set in **entropy_threshold** to eliminate noisy alerts with a lower entropy value.**none**: Do not use entropy thresholds. Cookbook will not filter out any alerts based on their entropy value.If you do not specify an entropy threshold, the default is global. The default global entropy threshold is 0. This means that unless you actively set up a global threshold, Cookbook will not filter out any alerts based on entropy values.See Configure Entropy Thresholds for more information on setting global and manager-specific entropy thresholds.
cook_for	Integer	No	Minimum time period, in seconds, that Cookbook clusters alerts for before the Recipe resets and starts a new cluster. See Cookbook and Recipe Examples If you set a different **cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for** time inherit the value from the Cookbook.
cook_for_extension	Integer	No	Time period that Cookbook can extend clustering

			alerts for before the Recipe resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Cookbook. As Cookbook receives related alerts, it continues to extend the total clustering time until the **max_cook_for** period is reached. Used in conjunction with the **max_cook_for** value, the **cook_for_extension** period helps to ensure that Cookbook continues to cluster alerts together that are related to the same failure. The **cook_for_extension** period only applies to new related alerts; it does not apply to existing alerts that are updated with new events. See Cookbook and Recipe Examples If you set a different **cook_for_extension** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for_extension** time inherit the value from the Cookbook.
max_cook_for	Integer	No	Maximum time period that Cookbook clusters alerts for before the Recipe resets and starts a new cluster. It works in conjunction with the **cook_for_extension** time to help ensure that Cookbook continues to cluster alerts together that are related to the same failure. This value is ignored unless the **cook_for_extension** time is specified. If **cook_for_extension** is set and this value is not set, the default is three times the **cook_for** value. See Cookbook and Recipe Examples If you set a different **max_cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **max_cook_for** value inherit the value from the Cookbook.
scale_by_severity	Boolean	No	Determines whether Cookbook ignores alerts with a severity of 0 (Clear). Set to **true** if you want Cookbook to ignore alerts with a severity of 0 (Clear). Set to **false** if you want Cookbook to include alerts with a severity of 0 (Clear).Default is **false**.
first_recipe_match_only	Boolean	No	Defines whether Cookbook treats Recipes in priority order. If set to **true**, Cookbook adds an alert to a cluster created by the highest priority Recipe that meets the clustering criteria. The priority order is defined by the order of the Recipes in the **recipes** list. If set to **false**, Cookbook adds an alert to clusters in all the Recipes that meet the clustering criteria. Default is **false**.
recipes	List of	Yes	A list of the Recipes in this Cookbook. You must supply at least one Recipe. If you set

	Strings		**first_recipe_match_only** to **first_match**, Cookbook uses the order of the Recipes in this list to determine their priority. The first Recipe has the highest priority.
run_on_startup	Boolean	No	Whether Cookbook should start when Moogfarmd starts. Default is **true**.
moobot	String	No	The Moobot you want Cookbook to use if there are any Bot Recipes. See Recipe Types for more information. Default is **Cookbook.js**.

Response

Endpoint **addCookbook** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

This endpoint returns an error code if the values of **entropy_threshold** and **threshold_type** are inconsistent. For example, if the **entropy_threshold** is set to 0.4 and **threshold_type** is set to global.

Successful requests return a JSON object containing the following:

Name	Type	Description
id	Integer	ID of the new Cookbook.

Examples

The following examples demonstrate typical use of endpoint **addCookbook**:

Request examples

Example cURL request to create a new Cookbook "GrazeCookBook6" using an entropy threshold with an explicit value for this Cookbook of 0.25:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addCookbook" -H
"Content-Type: application/json; charset=UTF-8" -d \
'{ \
    "name": "GrazeCookBook6", \
    "process_output_of": ["Alert Workflows"], \
    "recipes": ["Description","Source"], \
    "run_on_startup":false, \
    "first_recipe_match_only":true}, \
    "entropy_threshold" : 0.25, \
    "threshold_type": "explicit_value" \
}'
```

Example cURL request to create a new Cookbook "GrazeCookBook7" using entropy thresholds set up for individual managers.

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addCookbook" -H
"Content-Type: application/json; charset=UTF-8" -d \
'{ \
    "name": "GrazeCookBook7", \
    "process_output_of": ["Alert Workflows"], \
    "recipes": ["GrazeRecipe1","GrazeRecipe2","GrazeRecipe3"], \
    "run_on_startup":false, \
    "first_recipe_match_only":true}, \
```

```
    "threshold_type": "manager" \
}'
```

Response example

Successful response providing the ID of the new Cookbook that has been created:

```
{
    "id": 2
}
```

addEventsAnalyserWord

A Graze API POST request that adds a single word to an existing list of priority words or stop words in the Events Analyser configuration. This endpoint adds the word to the priority word list or stop word list depending on the argument you supply. Use updateEventsAnalyserWords to replace an entire list of priority words or stop words, or removeEventsAnalyserWord to remove a single word from a list of priority words or stop words.

See updateEventsAnalyserConfig to update the other fields in the Events Analyser configuration.

This endpoint adds a word to a list of priority words or stop words in the default partition only. See updateEventsAnalyserPartitionOverrides to configure priority words or stop words in partitions.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addEventsAnalyserWord** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
type	String	Yes	Determines whether the endpoint adds the word to the list of stop words or priority words. Set to **priority_word** to add to the list of priority words. Set to **stop_word** to add to the list of stop words.
word	String	Yes	Stop word or priority word that you want to add to the list.

Response

Endpoint **addEventsAnalyserWord** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **addEventsAnalyserWord**:

Request examples

Example cURL request to add the word 'fail' to the list of priority words:

```
curl -POST -u graze:graze -k -v
"https://localhost/graze/v1/addEventsAnalyserWord" \
--data-urlencode 'type=priority_word' \
--data-urlencode 'word="fail"'
```

Example cURL request to add the word 'maybe' to the list of stop words:

```
curl -POST -u graze:graze -k -v
"https://localhost/graze/v1/addEventsAnalyserWord" \
--data-urlencode 'type=stop_word' \
--data-urlencode 'word="maybe"'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addMergeGroup

A Graze API POST request that adds a new custom merge group.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addMergeGroup** takes the following request arguments:

Name	Type	Required	Description
name	String	Yes	A unique name for the custom merge group.
moolets	Array of Strings	Yes	List of clustering algorithm Moolets to include in the custom merge group.
alert_threshold	Integer	No	Minimum number of alerts that must be present in a cluster before it can become a Situation. Must be greater than or equal to 1. Enter **null** if you want the custom merge group to use the default merge group value. Default merge group value is 2.
situation_similarity_limit	Floating Point	No	Percentage of alerts two Situations must share before they are merged for this group. Enter a value between 0 and 1. Enter **null** if you want the merge group to use the default merge group value.

Response

Endpoint **addMergeGroup** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **addMergeGroup**:

Request example

Example cURL request to create a new custom merge group:

```
curl -X POST
-u graze:graze
-k -v "https://example.com/graze/v1/addMergeGroup"
-H "Content-Type: application/json; charset=UTF-8"
-d '{"name":"Merge Group 1","moolets":["Time Based (Tempus)", "Recipe
2"],"alert_threshold":2,"situation_similarity_limit":0.6}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addProcess

A Graze API POST request that adds a new process to the database. Processes are external business entities related to business activities that are affected by the incidents that Cisco Crosswork Situation Manager captures in Situations.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addProcess** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Process name.
description	String	No	Process description.

Response

Endpoint **addProcess** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **addProcess**:

Request example

Example cURL request to add a new process "New Proc 1" with a description:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addProcess" -H
"Content-Type: application/json; charset=UTF-8" -d '{"name" : "New Proc 1",
"description" : "This is my description 12345"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addService

A Graze API POST request that adds a new external service to the database. An external service is a business entity monitored by Moogsoft AIOps via event streams.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addService** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the external service you are adding.
description	String	No	Service description.

Response

Endpoint **addService** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **addService**:

Request example

Example cURL request to add service "New Service 1" with a description:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addService" -H
"Content-Type: application/json; charset=UTF-8" -d '{"name" : "New Service 1",
"description" : "This is my description 12345"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addSigCorrelationInfo

A Graze API POST request that associates the external client with a specified Situation. This allows Cisco Crosswork Situation Manager to filter events and send only those of interest to an external system.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addSigCorrelationInfo** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.

service_name	String		Name of the external service, for example, ServiceNow.
resource_id	String		ID of the external service entity to associate with this Situation.

Response

Endpoint **addSigCorrelationInfo** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **addSigCorrelationInfo**:

Request example

Example cURL request to associate resource ID "my resource 7" in service "my service 7" with Situation ID 7:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/addSigCorrelationInfo" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 3, "service_name" : "my
service 7", "resource_id" : "my resource 7"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addSituationCustomInfo

A Graze API POST request that adds and merges custom information for a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addSituationCustomInfo** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request.

			See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.
custom_info	JSON Object	Yes	A JSON object containing the custom information.

Response

Endpoint **addSituationCustomInfo** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **addSituationCustomInfo**:

Request example

Example cURL request to add custom info to "field1", "field2", "field3", and "field4" in Situation ID 5:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/addSituationCustomInfo" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 5, "custom_info" : { "field1"
: "value1" , "field2" : "value2" , "field3" : ["item1","item2","item3"] ,
"field4" : {"field4-1" : "value4-1","field4-2" : "value4-2"} }}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addTempus

A Graze API POST request that adds a new Tempus Moolet.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addTempus** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more

			information.
name	String	Yes	Name of the Tempus algorithm. Must be unique.
description	String	No	Description of the Situations Tempus generates. Default is 'A Tempus Situation'.
entropy_threshold	Number	No	Minimum entropy value an alert must have for Tempus to consider it for clustering into a Situation. A value between 0 and 1. Only relevant if **threshold_type** is set to **explicit_value**. If used, Tempus does not cluster any alerts with an entropy value below the threshold into Situations. Default is 0.0 which means that Tempus processes all alerts.
threshold_type	String	No	Type of entropy threshold you want Tempus to use. One of:**global**: Use the global entropy threshold. This is a single entropy threshold that Tempus applies to all alerts to eliminate noisy alerts with a lower entropy value.**manager**: Use entropy thresholds set up for individual managers. Tempus uses this value to eliminate noisy alerts with a lower entropy value. If an alert's manager does not have an entropy threshold, Tempus uses the global entropy threshold to filter out alerts.**explicit_value**: Use the value set in **entropy_threshold** to eliminate noisy alerts with a lower entropy value.**none**: Do not use entropy thresholds. Tempus will not filter out any alerts based on their entropy value.If you do not specify an entropy threshold, the default is global. The default global entropy threshold is 0. This means that unless you actively set up a global threshold, Tempus will not filter out any alerts based on entropy values.See Configure Entropy Thresholds for more information on setting global and manager-specific entropy thresholds.
execution_interval	Number	No	Executes Tempus after a defined number of seconds. Default is 120.
window_size	Number	No	Determines the length of time, in seconds, when Tempus analyzes alerts and clusters them into a Situation each time it runs. Default window size is 1200 seconds (20 minutes). The default window size and bucket size provides 240 buckets per time period.
bucket_size	Number	No	Determines the time span, in seconds, of each bucket in which alerts are captured. Default bucket size is 5 seconds. The default window size and bucket size provides 240 buckets per time period.
arrival_spread	Number	No	Sets the acceptable latency or arrival window for each alert, in seconds. Use this to minimise or reduce the impact of multiple alerts arriving over a small amount

			of time and landing in separate buckets. This is a value between 1 and 60. Default is 15.
minimum_arrival_similarity	Number	No	How similar alerts must be for Tempus to consider them for clustering. Default is 0.6667.
alert_threshold	Number	No	Minimum number of alerts that match the clustering criteria before the Tempus algorithm creates a Situation. Default is 4. When Tempus determines the number of alerts required to create a Situation, it compares the alert threshold values in Tempus and in the merge group that Tempus belongs to, and it uses the higher value. If you are using the default merge group which has an alert threshold of 2, Tempus will never create a Situation containing a single alert. If you want Cisco Crosswork Situation Manager to create Situations with a single alert, consider changing the alert threshold in the default merge group to 1 or creating custom merge groups. See Merge Groups for more information on updating the default merge group and setting up custom merge groups.
process_output_of	Array of Strings	Yes	Defines the source of the alerts that Tempus processes. You can specify none, one or more Moolets. Typically Tempus processes the output of its direct upstream neighbor in the processing chain. Usually this is "Alert Workflows" which are the output from the Alert Workflow Engine.
run_on_startup	Boolean	No	Whether Tempus should start when Moogfarmd starts. Default is **true**.
partition_by	String	No	Splits clustering according to the entered component. After alerts have been clustered and before they enter merging and resolution, you can split clusters into sub-clusters based on a component of the events. For example, you can use the **manager** parameter to ensure that Situations only contain events from the same manager. The default of null means that no partitioning occurs. Note Cisco does not recommend partitioning by components.
pre_partition	Boolean	No	Partitions event streams before clustering. You specify a component field on which the event stream will be partitioned before clustering occurs. The alerts in the resulting Situations each contain a single value for the component field chosen. The default of null means that no pre-partitioning occurs.
significance_test	String	No	Calculation that determines how significant a cluster of alerts or a potential Situation must be for Tempus to

			detect it. The default, **Poisson1**, looks at the data of a single alert cluster to calculate how significant it is. The default is more likely to detect all significant alert clusters but with a higher risk of creating insignificant alert clusters. Use this option when your alerts originate from different networks or unrelated topologies. **Poisson2** is a more thorough test that looks at an alert cluster and all alerts outside the cluster with a similar event rate. It is more likely to exclude all insignificant alert clusters but with a high risk of excluding significant alert clusters. Use this option if you expect all of your alerts to come from the same connected network. See Poisson distribution for more information.
significance_threshold	Number	No	Sets the maximum significance score for Tempus to create a Situation. The score is proportional to the probability that the alert cluster or potential Situation was coincidence. The lower the score, the more significant the cluster and the least likely it was a coincidence. This score ranges from 0 to 100. Default is 1.
detection_algorithm	String	No	Detection algorithm that Tempus uses, one of: **Louvain**, **LouvainMulti**, or **SmartLocal**. Default is **Louvain**.

Response

Endpoint **addTempus** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

This endpoint returns an error code if the values of **entropy_threshold** and **threshold_type** are inconsistent. For example, if the **entropy_threshold** is set to 0.4 and **threshold_type** is set to global.

Examples

The following examples demonstrate typical use of endpoint **addTempus**:

Request example

Example cURL request to create a new Tempus algorithm:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addTempus" -H
"Content-Type: application/json; charset=UTF-8" -d \
'{ \
"name":"GrazeTempus1", \
"description":"Situation Generated by Tempus", \
"process_output_of":"Alert Workflows", \
"run_on_startup":false, \
"entropy_threshold":0.3, \
```

```
"threshold_type":"explicit_value", \
"execution_interval":60, \
"window_size":240, \
"bucket_size":3, \
"arrival_spread":9, \
"minimum_arrival_similarity":0.5, \
"alert_threshold":5, \
"partition_by":"manager", \
"significance_test":"Poisson2", \
"significance_threshold":3, \
"detection_algorithm":"LouvainMulti" \
}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

addThreadEntry

A Graze API POST request that adds a new entry to an existing thread in a Situation. Threads are comments or 'story activity' on Situations.

This endpoint returns the entry ID of the newly created thread entry.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **addThreadEntry** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.
thread_name	String	Yes	Name of the existing thread.
entry	String	Yes	Description of the new entry you want to create in the thread. For example, "And another thing...". HTML and XML tags are stripped from the thread entry text. Reserved characters are converted to HTML entities, for example, **&** is converted to **&**.
resolving_step	Boolean	No	Whether or not the thread entry you are adding is a resolving step.

Response

Endpoint **addThreadEntry** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
entry_id	Number	ID of the new thread entry.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`addThreadEntry`**:

Request example

Example cURL request to add a new entry "Test Entry" to thread "Support" in Situation 3:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addThreadEntry" -H
"Content-Type: application/json; charset=UTF-8" -d '{"sitn_id" : 3,
"thread_name" : "Support", "entry" : "Test Entry", "resolving_step" : true}'
```

Response example

Successful response providing the ID of the thread entry that has been created:

```
{"entry_id":27}
```

addValueRecipe

A Graze API POST request that creates a new Cookbook Recipe using Value Recipe or Value Recipe v2 recipe types. See Recipe Types.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`addValueRecipe`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`cookbook`	List of Strings	No	A list of the Cookbooks that this Recipe belongs to. You can add Cookbooks here or, when you create a Cookbook, you can assign the Recipes to it.
`name`	String	Yes	Name of the Recipe. Use a unique and descriptive name.
`description`	String	No	Description of the Recipe. Default is the Recipe **`name`**.

version	String	No	Defines whether the Recipe uses Value Recipe or Value Recipe v2. Valid values are **V1** for the Value Recipe and **V2** for Value Recipe v2. Default is **V2**. See Recipe Types for more information. Use addBotRecipe if you want to create a Bot Recipe.
alert_threshold	Positive Integer	No	Minimum number of alerts required before Cookbook creates a Situation. When Cookbook determines the number of alerts required to create a Situation, it compares the alert threshold values in the Cookbook Recipe and in the merge group that the Cookbook Recipe belongs to, and it uses the higher value. If you are using the default merge group which has an alert threshold of 2, Cookbook will never create a Situation containing a single alert. If you want Cisco Crosswork Situation Manager to create Situations with a single alert, consider changing the alert threshold in the default merge group to 1 or creating custom merge groups. See Merge Groups for more information on updating the default merge group and setting up custom merge groups.
trigger	String	No	A filter that determines the alerts that Cookbook considers for Situation creation. Cookbook includes alerts that match the trigger filter. By default Cookbook only includes alerts with a severity of 'Critical'.
exclusion	String	No	A filter that determines the alerts to exclude from Situation creation. Cookbook ignores alerts that match the exclusion filter
seed_alert	String	No	A filter that determines whether to create a Situation from a seed alert. The seed alert must meet both the **trigger**, **exclusion** and **seed_alert** criteria to create a Situation. Cookbook considers subsequent alerts for clustering if they meet the trigger and exclusion filter criteria. Alerts that arrive prior to the seed alert that met the trigger and exclusion filter criteria do not form Situations.
rate	Double	No	Rate, in number of alerts per second. Cookbook clusters alerts if they arrive at a higher rate than is specified here. Cookbook uses **rate** together with **min_sample_size** and **max_sample_size** to determines whether to cluster alerts into Situations. See Cookbook and Recipe Examples.
min_sample_size	Positive Integer	No	Number of alerts that must arrive before the Cookbook starts to calculate the alert rate. See Cookbook and Recipe Examples for more information. Default is 5. Valid only if **rate** is non-zero.

max_sample_size	Positive Integer	No	Maximum number of alerts that are considered in the alert rate calculation. When more than this number of alerts have arrived, Cookbook discards the oldest alerts and calculates the alert rate based on the number of alerts in the **max_sample_size**. See Cookbook and Recipe Examples for more information. Default is 10. Valid only if **rate** is non-zero.
cook_for	Positive Integer	No	Minimum time period, in seconds, that the Cookbook Recipe clusters alerts for before it resets and starts a new cluster. See Cookbook and Recipe Examples. If you set a different **cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for** time inherit the value from the Cookbook. Inherits value from Cookbook if omitted.
cook_for_extension	Positive Integer	No	Time period that the Cookbook Recipe can extend clustering alerts for before it resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Cookbook. As Cookbook receives related alerts, it continues to extend the total clustering time until the **max_cook_for** period is reached. Used in conjunction with the **max_cook_for** value, the **cook_for_extension** period helps to ensure that Cookbook continues to cluster alerts together that are related to the same failure. The **cook_for_extension** period only applies to new related alerts; it does not apply to existing alerts that are updated with new events. Cookbook and Recipe Examples. If you set a different **cook_for_extension** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for_extension** time inherit the value from the Cookbook. Inherits value from Cookbook if omitted.
max_cook_for	Positive Integer	No	Maximum time period that the Cookbook Recipe clusters alerts for before it resets and starts a new cluster. It works in conjunction with the **cook_for_extension** time to help ensure that Cookbook continues to cluster alerts together that are related to the same failure. This value is ignored unless the **cook_for_extension** time is specified. Cookbook and Recipe Examples. If you set a different **max_cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **max_cook_for** value inherit the value from the

			Cookbook. Inherits value from Cookbook if omitted.
cluster_by	String	No	Determines Cookbook's clustering behavior. Set to an empty string to use the Cookbook **cluster_by** setting. Set to **first_match** so that Cookbook adds alerts to the first cluster over the similarity threshold value. Set to **closest_match** to add alerts to the cluster with the highest similarity greater than the similarity threshold value. This option may be less efficient because Cookbook needs to compare alerts against each cluster in a Recipe. Default is an empty string which means the Recipe uses the Cookbook setting. If you set a different **cluster_by** value for a Recipe, it overrides the Cookbook value. Recipes without a **cluster_by** value inherit the value from the Cookbook.
hop_limit	Positive Integer	No	Maximum number of hops between the alert source nodes in order for the alerts to quality for clustering. Cisco Crosswork Situation Manager measures hop limit from the first alert that formed the Situation and always follows the shortest possible route. A hop is the distance between two directly connected nodes. You can only set a hop limit if you have one or more topologies in your system. For more information on hops and hop limit see Vertex Entropy and Configure Topology-based Clustering with Vertex Entropy. For more information on topologies see Topologies.
components	JSON Array	Yes	Values that alerts must match for Cookbook to include them in a Situation. You can provide values for multiple components. See the table below for a full description of all components.
use_dynamic_topology	Boolean	No	Infer the topology to cluster on from the **moog_topology** field in the alert's custom info. If you use a dynamic topology you cannot set **topology_name**.
alert_matching_attribute	String	No	The alert field that specifies the topology node from which the alert was generated. If you set an alert matching attribute you must set **dynamic_topology** to **true** or set the **topology_name**.
topology_name	String	No	Restrict clustering to nodes in the specified topology. If you set a topology name you cannot set **dynamic_topology** to **true**.

The **components** property is an array of JSON objects containing the following:

Name	Type	Required	Description
name	String	Yes	Name of the component.
similarity	Double	Yes	Similarity threshold that the component must meet for Cookbook to cluster the alert into a Situation.
shingle_size	Integer	No	Shingle size for Cookbook to use to determine the similarity between different strings. The shingle size is only valid for Recipe Value v2 recipes. Default is -1 which means that Cookbook uses words to determine similarity. See Recipe Types.
treat_as	String	No	Determines whether Cookbook treats the component as a string or matches each value in the list individually. See Recipe Types for details. Valid values are **List** and **String**. Default is **String**.
case_sensitive	Boolean	No	Enables or disables case sensitive when comparing strings. Case sensitivity is only valid for Recipe Value recipes. See Recipe Types more details. Default is **true** which means that strings are treated as case sensitive.

Response

Endpoint **addValueRecipe** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
id	Integer	ID of the new Value Recipe.

Examples

The following examples demonstrate typical use of endpoint **addValueRecipe**:

Request example

Example cURL request to add a new Recipe "GrazeRecipe":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addValueRecipe" -H
"Content-Type: application/json; charset=UTF-8" -d '{"cookbook" : "GrazeCook1",
"name": "GrazeRecipe", "alert_threshold" : 1,
"hop_limit" : 0,
"dynamic_topology" : false,
"topology_name": "physical",
"components" :  [{
    "name": "custom_1",
    "similarity": 0.2,
    "shingle_size": 2 }]
}'
```

Example cURL request to add a new Recipe "GrazeRecipe2":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/addValueRecipe" -H
"Content-Type: application/json; charset=UTF-8" -d '{"cookbook" : "GrazeCook1",
"name": "GrazeRecipe2", "alert_threshold" : 1,
"hop_limit" : 0,
"dynamic_topology" : true,
"alert_matching_attribute" : "host",
"components" :  [{
    "name": "custom_1",
    "similarity": 0.2,
    "shingle_size": 2 } ]
}'
```

Response example

Successful response providing the ID of the new Value Recipe that has been created:

```
{
    "id": 6
}
```

applyNewLicense

A Graze API POST request that adds a Cisco Crosswork Situation Manager license via Graze.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`applyNewLicense`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`license`	String	Yes	A valid license key.

Response

Endpoint **`applyNewLicense`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **`applyNewLicense`**:

Request example

Example cURL request to add a valid license:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/applyNewLicence" -
H "Content-Type: application/json; charset=UTF-8" -d '{"license" : "<your
license key>"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

assignAlert

A Graze API POST request that assigns the specified user as the owner of the specified alert ID.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **assignAlert** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
alert_id	Number	Yes	Alert ID.
user_id	Number	No, if you specify **username**.	ID of the user to be assigned as the owner of the alert. You must provide the **user_id** or the **username**.
username	String	No, if you specify **user_id**.	Username of the user to be assigned as the owner of the alert. You must provide the **user_id** or the **username**.

Response

Endpoint **assignAlert** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **assignAlert**:

Request example

Example cURL request to username "network1" to alert ID 7:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/assignAlert" -H
"Content-Type: application/json; charset=UTF-8" -d '{"alert_id" : 7, "username"
: "network1" }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

assignAndAcknowledgeAlert

A Graze API POST request that assigns and acknowledges the specified user as the owner of the specified alert ID.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`assignAndAcknowledgeAlert`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`alert_id`	Number	Yes	Alert ID.
`user_id`	Number	No, if you specify **`username`**.	ID of the user to be assigned as the owner of the alert. You must provide the **`user_id`** or the **`username`**.
`username`	String	No, if you specify **`user_id`**.	Username of the user to be assigned as the owner of the alert. You must provide the **`user_id`** or the **`username`**.

Response

Endpoint **`assignAndAcknowledgeAlert`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`assignAndAcknowledgeAlert`**:

Request example

Example cURL request to assign user "Cloud DevOps 1" as the owner of alert 432:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/assignAndAcknowledgeAlert" -H "Content-Type:
```

```
application/json; charset=UTF-8" -d '{"alert_id" : 432, "username" : "Cloud
DevOps 1" }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

assignAndAcknowledgeSituation

A Graze API POST request that assigns and acknowledges the moderator to the Situation for a specified situation ID and user ID.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`assignAndAcknowledgeSituation`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.
`user_id`	Number	Yes	ID of the user to be assigned as the owner of the Situation.
`username`	String	No	Username of the user to be assigned to the Situation.

Response

Endpoint **`assignAndAcknowledgeSituation`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`assignAndAcknowledgeSituation`**:

Request example

Example cURL request to set user ID 7 as the moderator on Situation ID 975 and acknowledge this:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/assignAndAcknowledgeSituation" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 975, "user_id" : 7 }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

assignSituation

A Graze API POST request that assigns the moderator to the Situation for a specified Situation ID and user ID.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **assignSituation** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.
user_id	Number	Yes	User ID.
username	String	No	A valid username.

Response

Endpoint **assignSituation** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **assignSituation**:

Request example

Example cURL request:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/assignSituation" -
H "Content-Type: application/json; charset=UTF-8" -d '{"sitn_id" : 7, "user_id"
: 3 }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

assignTeamsToSituation

A Graze API POST request that assigns one or more teams to a Situation. Once successfully run, Cisco Crosswork Situation Manager marks the Situation as overridden and the Teams Manager Moolet can no longer modify its team assignment. See Teams Manager Moolet for more information.

This endpoint replaces any teams previously assigned to the Situation. You can also use it to deassign all teams from a Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **assignTeamsToSituation** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.
team_ids	List	No, if you specify **team_names**.	A list of team IDs to assign to the Situation. Specify an empty list to deassign all teams from the Situation.
team_names	List	No, if you specify **team_ids**.	A list of team names to assign to the Situation. Specify an empty list to deassign all teams from the Situation.

Response

Endpoint **assignTeamsToSituation** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing one of the following, depending on the request argument used:

Name	Type	Description
team_ids	List	A list of team IDs assigned to the Situation.
team_names	List	A list of team names assigned to the Situation.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`assignTeamsToSituation`**:

Example assigning team IDs

Example cURL request to assign team IDs 1 and 2 to Situation 1:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/assignTeamsToSituation" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 1 , "team_ids" : [ 1, 2 ]}'
```

Example response returning the team IDs (1 and 2) that have been successfully assigned to the Situation:

```
{"team_ids" : [ 1,2 ]}
```

Example assigning team names

Example cURL request to assign teams "Network_US" and "Network_UK" to Situation 2:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/assignTeamsToSituation" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 2 , "team_names" : [
"Network_US", "Network_UK" ]'
```

Example response returning the team names ("Network_US" and "Network_UK") that have been successfully assigned to the Situation:

```
{"team_names" : [ "Network_US", "Network_UK" ]}
```

Example unassigning teams

Example cURL request to unassign all teams from Situation 1:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/assignTeamsToSituation" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 1 , "team_ids" : []}'
```

Example response returning an empty list showing that all the teams have been successfully unassigned from the Situation:

```
{"team_ids" : []}
```

authenticate

A Graze API GET request that provides the auth_token required by all other Graze API requests which do not provide the basic authentication header. Graze users can have multiple concurrent Graze sessions with the same username and password.

All requests (other than **authenticate**) require a valid **auth_token** or basic authentication header. Therefore before any Graze API request is used, a valid **authenticate** request must be successfully made unless basic authentication headers are used.

Inactive sessions will be logged out after one hour, and a new **authenticate** request must be made to get a new valid **auth_token**.

If you make regular Graze requests within a one hour timeframe, you are considered active and your session does not expire. Inactive sessions are logged out after one hour, and you must make a new **authenticate** request to get a new valid **auth_token**.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **authenticate** takes the following request arguments:

Name	Type	Required	Description
username	String	Yes	A valid Cisco Crosswork Situation Manager username.
password	String	Yes	The username's corresponding password.

Response

Endpoint **authenticate** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
auth_token	String	A session ID for use in subsequent requests.

Examples

The following examples demonstrate typical use of endpoint **authenticate**:

Request examples

Example cURL request to return an authorization token for username "JohnJones" and password "password":

```
curl -k -v
"https://localhost/graze/v1/authenticate?username=JohnJones&password=password"
```

Example cURL request to return an authorization token for the Graze user:

```
curl -k -v
"https://localhost/graze/v1/authenticate?username=graze&password=graze"
```

Response example

Example response returning the authorization token:

```
{"auth_token":"878b3ec57d464aee80d09893221be8e8"}
```

checkSituationFlag

A Graze API GET request that checks whether a flag is associated with a Situation.

See Situation Flags for more information on Cisco Crosswork Situation Manager Situation flags.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **checkSituationFlag** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	ID of the Situation to check.
flag	String	Yes	Name of the flag to check for the specified Situation ID.

Response

Endpoint **checkSituationFlag** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Type	Description
Boolean	Whether or not the flag is associated with the specified Situation.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **checkSituationFlag**:

Request example

Example cURL request to request a Boolean whether the specified Situation has the flag "NOTIFIED" associated with it.

```
curl -X GET -u graze:graze -k -v
https://localhost/graze/v1/checkSituationFlag?sitn_id=1&flag=NOTIFIED
```

Response example

Example response returning the Boolean value **`true`** because the Situation contains the specified flag:

```
true
```

closeAlert

A Graze API POST request that closes one or more alerts.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`closeAlert`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`alert_id`	Number	No, if you specify **`alert_ids`**.	A single alert ID. You must provide a single **`alert_id`** or a list of **`alert_ids`**.
`alert_ids`	Array of Numbers	No, if you specify **`alert_id`**.	A list of alert IDs. You must provide a single **`alert_id`** or a list of **`alert_ids`**.
`thread_entry_comment`	String	No	Thread entry comment you want to add to the closed alert. HTML and XML tags are stripped from the thread entry text. Reserved characters are converted to HTML entities, for example, **`&`** is converted to **`&`**.

Response

Endpoint **`closeAlert`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`closeAlert`**:

Request example

Example cURL request to close alert IDs 78, 234, and 737:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/closeAlert" -H
"Content-Type: application/json; charset=UTF-8" -d '{"alert_ids" :
[78,234,737],"thread_entry_comment" : "Closing as agreed during team discussion
1/1/2018" }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

closeSituation

A Graze API POST request that closes a specified Situation which is currently open, and optionally closes alerts in the Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`closeSituation`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.
`resolution`	Number	Yes	Determines what to do with the alerts in the Situation: 0 = Close no alerts. 1 = Close all alerts in this Situation. 2 = Close only alerts unique to this Situation.

Response

Endpoint **`closeSituation`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes

Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`closeSituation`**:

Request example

Example cURL request to close Situation 7 and leave all its alerts open:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/closeSituation" -H
"Content-Type: application/json; charset=UTF-8" -d '{"sitn_id" : 7, "resolution"
: 0 }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

createMaintenanceWindow

A Graze API POST request that creates a maintenance window. A maintenance window filters alerts caused by a known period of maintenance.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`createMaintenanceWindow`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`name`	String	Yes	Name of the maintenance window.
`description`	String	Yes	Description of the maintenance window.
`filter`	String	Yes	SQL-like filter that alerts must match to be included in the maintenance window. If the filter includes a backslash in the filter string, you need to double escape these to maintain the backslash character in the filter string. However, if you have **`\t`**, **`\n`**, **`\b`**, or **`\r`** followed by a backslash in the path, you do not need to pass any extra backslashes. See the Example of escaped characters.
`start_date_time`	Number	Yes	Start time of the maintenance window. This must be in Unix epoch time and may be up to 5 years in the future.
`duration`	Number	Yes	Duration of the maintenance window in seconds. The minimum duration is 1 second and the maximum is

			157784630 seconds (5 years).
forward_alerts	Boolean	Yes	Whether or not alerts should be forwarded to the next Moolet in the processing chain.
recurring_period	Number	No	Whether or not this is a recurring maintenance window. Set this to:**1** for a recurring maintenance window.**0** for a one-time maintenance window.If not specified, default is **0**. If you set this property to **1**, you must specify **recurring_period_units**.
recurring_period_units	Number	No	Specifies the recurring period of the maintenance window, in days, weeks or months. Valid values are:**2** = daily**3** = weekly**4** = monthlyDefault is **0** if **recurring_period** is set to **0**.
timezone	String	No	Time zone that you want the maintenance window to be created in. Default is the time zone of the user that makes the request. If the user has a "SYSTEM" time zone, Cisco Crosswork Situation Manager uses the MoogSvr time zone. The time zone must be a valid entry in the IANA Time Zone Database. When scheduling recurring maintenance windows, Cisco Crosswork Situation Manager takes into account any daylight savings time changes for the time zone.

Response

Endpoint **createMaintenanceWindow** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
window_id	Number	ID of the new maintenance window.

Examples

The following examples demonstrate typical use of endpoint **createMaintenanceWindow**:

Request examples

Example cURL request to create a window, which recurs once a month (from its start_date_time), with a filter where the source is "server1" and the external ID is one of "value1", "value2", or "value3":

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/createMaintenanceWindow" -H "Content-Type:
application/json; charset=UTF-8" -d '{"name":"window1", "description":"window1
description here", "filter": "source = \"server1\" and external_id in
(\"value1\", \"value2\", \"value3\")", "start_date_time": 1473849237,
"duration": 55800, "forward_alerts": false, "recurring_period": 1,
"recurring_period_units": 4}'
```

Example cURL request to create a one-time maintenance window, which is filtered where the source is equal to "hostIsDown":

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/createMaintenanceWindow" -H "Content-Type:
application/json; charset=UTF-8" -d '{"name": "my_window_1", "description":
"This is my description", "filter": "source = \"hostIsDown\"",
"start_date_time": 1473849237, "duration": 55800, "forward_alerts": false}'
```

Example cURL request to create a daily maintenance window in the "America/New York" time zone:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/createMaintenanceWindow" -H "Content-Type:
application/json; charset=UTF-8" -d '{"name":"window", "description":"window
with specified timezone", "filter": "source = \"server1\" and external_id in
(\"value1\", \"value2\", \"value3\")", "start_date_time": 1564566188,
"duration": 3600, "forward_alerts": false, "recurring_period": 1,
"recurring_period_units": 2, "timezone": "America/New_York"}'
```

Example of escaped characters

Example cURL request using multiple escaped backslash characters in the filter to maintain the correct characters in the filter string:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/createMaintenanceWindow" -H "plication/json;
charset=UTF-8" -d '{"name":"LON Server 1", "description":"[10453] ",
"filter":"NOT (alert_id IS NULL) AND (agent_location MATCHES \"LON_S1\") AND
(custom_info.eventDetails.alert_customer MATCHES \"LON\") AND (manager MATCHES
\"LOGFILE\") AND (class MATCHES \"could not execute backup\") AND
(custom_info.eventDetails.field_1 MATCHES
\"C:\\\\\\\\\\\LON_SVR1\\Backup\\2019\")", "start_date_time":1480483478,
"duration":86400, "recurring_period_units":2, "recurring_period":1,
"forward_alerts":false}'
```

The endpoint **`createMaintenanceWindow`** creates the correct filter:

```
((((( NOT (alert_id IS NULL)) AND (`Agent location` MATCHES "LON_S1")) AND
(custom_info.eventDetails.alert_customer MATCHES "LON")) AND (Manager MATCHES
"LOGFILE")) AND (Class MATCHES "could not execute backup")) AND
(custom_info.eventDetails.field_1 MATCHES "C:\\\\LON_SVR1\Backup\2019")
```

Response example

A successful request returns the ID of the new maintenance window:

```
{
    "window_id":16
}
```

createSecurityRealm

A Graze API POST request that creates a new SAML security realm from an Identity Provider (IdP) URL. The request also adds the realm configuration you provide.

Warning

Warn any users who are logged into Cisco Crosswork Situation Manager using the default realm before using this request. Cisco Crosswork Situation Manager may log out users when the new realm becomes active.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **createSecurityRealm** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the security realm.
type	String	Yes	Security realm type. This must be **"SAML2"**.
active	Boolean	Yes	Determines whether the new realm is active in Cisco Crosswork Situation Manager on creation. You can create an inactive realm for testing purposes. For example, you can verify if a security realm with that name already exists or if it fails.
configuration	JSON Object	Yes	JSON object containing the realm configuration. For information on the configuration properties, see Security Configuration Reference. Upload your IdP metadata file using idpMetadata or specify the location of the file using idpMetadataUrl. For example: **"idpMetadataUrl":"http://<location_of_idp_metadata>"** **"idpMetadata":"<raw_ipd_metadata.xml>"**

Response

Endpoint **createSecurityRealm** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **createSecurityRealm**:

Request example

Example cURL request to create a security realm:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/createSecurityRealm" -d '{
    "name":"mySamlRealm",
    "type":"SAML2",
    "active:"true",
    "configuration":
    {

"idpMetadataUrl":"http://exampleIdP:18080/auth/realms/master/protocol/saml/descr
iptor",
            "defaultRoles":["Operator"],
            "defaultTeams":["Cloud DevOps"],
```

```
                "defaultGroup":"End-User",
                "existingUserMappingField":"username",
                "username":"$username",
                "email":"$email",
                "fullname":"$firstname $lastname",
                "maximumAuthenticationLifetime":60
        }
    }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

createSituation

A Graze API POST request that creates a manual Situation. The Situation description is set with the **`description`** parameter.

The following Situation settings are pre-set and not configurable here:

1. Status: Opened
2. Moderator: none assigned

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`createSituation`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`description`	String	Yes	Description of the new Situation.

Response

Endpoint **`createSituation`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`sitn_id`	Number	ID of the newly created Situation.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation

Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`createSituation`**:

Request example

Example cURL request to create a Situation with the description "Database Outage 08/06/2019":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/createSituation" -
H "Content-Type: application/json; charset=UTF-8" -d '{"description" : "Database
Outage 08/06/2019"}'
```

Response example

Example response returning the ID of the newly created Situation:

```
{"sitn_id":2300}
```

createTeam

A Graze API POST request that creates a new team.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`createTeam`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`name`	String	Yes	New team name. Must be unique.
`alert_filter`	String	No	An SQL-like filter that alerts must match to be assigned to the team.
`services`	Array of Strings or Numbers	No	List of the team service names or IDs.
`sig_filter`	String	No	An SQL-like filter that Situations must match to be assigned to the team.
`landing_page`	String	No	Team default landing page.
`active`	Boolean	No	False if the team is inactive, true if the team is active. Default is **`true`**.
`description`	String	No	Team description.
`users`	Array of Numbers or	No	Team users (either IDs or usernames).

	Strings		

Response

Endpoint **createTeam** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
team_id	Number	ID of the new team.

Examples

The following examples demonstrate typical use of endpoint **createTeam**:

Request example

Example cURL request to create a team called "my team name" consisting of user1, user2, and user3:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/createTeam" -H
"Content-Type: application/json; charset=UTF-8" -d '{"name" : "my team name 1",
"alert_filter" : "manager = \"my_manager\" and (class = \"my_class_12345\" or
external_id = \"my_ext_12345\")", "services" : ["Identity Management","Yellow
Pages"], "sig_filter" : "description = \"my_description_12345\" or queue = 50",
"landing_page" : {"type":"situations","id":"open"}, "active" : true,
"description" : "The team description 12345", "users" :
["user1","user2","user3"]}'
```

Response example

Example response returning the ID of the created team:

```
{"team_id":16}
```

createThread

A Graze API POST request that creates a new thread for a specified Situation. Threads are comments or 'story activity' on Situations.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **createThread** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	ID of the Situation you want to create a new thread for.
thread_name	String	Yes	Name of the new thread.

Response

Endpoint **createThread** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **createThread**:

Request example

Example cURL request to create a new thread "Thread 0958" on Situation ID 176:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/createThread" -H
"Content-Type: application/json; charset=UTF-8" -d '{"sitn_id" : 176,
"thread_name" : "Thread 0958"}
```

Response example

A successful request returns the HTTP code 200 and no response text.

createThreadEntry

Note

This endpoint has been superseded. Use addThreadEntry instead. All new functionality will be delivered in addThreadEntry.

A Graze API POST request that creates a new entry in an existing thread in a Situation. Threads are comments or 'story activity' on Situations.

This endpoint returns a Boolean indicating whether or not the thread entry was created successfully.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **createThreadEntry** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.

thread_name	String	Yes	Name of the existing thread.
entry	String	Yes	Description of the new entry you want to create in the thread. For example, "And another thing...". HTML and XML tags are stripped from the thread entry text. Reserved characters are converted to HTML entities, for example, **&** is converted to **&**.
resolving_step	Boolean	No	Whether or not the thread entry you are creating is a resolving step.

Response

Endpoint **createThreadEntry** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Type	Description
Boolean	Whether or not the new thread entry was created successfully.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **createThreadEntry**:

Request example

Example cURL request to create a new entry in thread "Support" in Situation 3:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/createThreadEntry"
-H "Content-Type: application/json; charset=UTF-8" -d '{"sitn_id" : 3,
"thread_name" : "Support", "entry" : "Test Entry", "resolving_step" : true}'
```

Response example

Example response showing that the new thread entry was successfully created::

```
true
```

createUser

A Graze API POST request that creates a new user.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **createUser** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
username	String	Yes	Login username for the new user. Must be unique.
password	String	Yes	New user password, only valid for DB realm.
active	Boolean	Yes	**true** if the user is active, **false** if the user is inactive. Default is **true**.
email	String	Yes	User's email address.
fullname	String	Yes	User's full name.
roles	Array of Strings or Numbers	Yes	List of either the role IDs or the role names. For example, **"roles":["Super User"]**.
primary_group	String or Number	Yes	User's primary group name or primary group ID.
department	String or Number	Yes	User's department ID or department name.
joined	Number	Yes	Time the user joined in Unix epoch time.
timezone	String	Yes	User's timezone.
contact_num	String	Yes	User's phone number.
session_expiry	Number	Yes	Number of minutes after which the user's session expires. Default is the system default.
teams	Array of Numbers or Strings	Yes	List of the user's team names or team IDs.

Response

Endpoint **createUser** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
user_id	Number	ID of the new user.

Examples

The following examples demonstrate typical use of endpoint **createUser**:

Request example

Example cURL request to create a new user "johndoe1":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/createUser" -H
"Content-Type: application/json; charset=UTF-8" -d '{"username" : "johndoe1",
"roles" : ["Super User", "Operator"], "password" : "johndoe1", "active" : true,
"email" : "johndoe@moogsoft.com", "fullname" : "John Doe", "primary_group" :
"Network", "department" : "Support", "joined" : 1494951621, "timezone" :
"Europe/London", "contact_num" : "555-1234", "session_expiry" : null, "teams" :
["my team 1","my team 2","my team 3"], "properties" : null}'
```

Response example

Example response returning the new user ID:

```
{"user_id":777}
```

createWorkflow

A Graze API POST request that creates a new workflow at the end of a Workflow Engine Moolet sequence. The new workflow is automatically active. To move it, use reorderWorkflows.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **createWorkflow** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
moolet_name	String	Yes	Name of the Workflow Engine Moolet that the new workflow belongs to.
workflow_name	String	Yes	Name of the new workflow. Must be unique.
description	String	No	Description of the new workflow.
entry_filter	JSON Object	No	An SQL-like filter to determine which events, alerts or Situations can enter the workflow. Leave empty for the workflow to accept all events, alerts or Situations.
sweep_up_filter	JSON Object	No	An SQL-like filter to intake any additional events, alerts or Situations from the database.
first_match_only	Boolean	Yes	If enabled, events, alerts, and Situations only pass through actions on the first time they enter this workflow.
operations	JSON	Yes	List of properties relating to each operation:

<table>
<tr>
<td></td>
<td>List</td>
<td></td>
<td>
<table>
<tr><th>Name</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>type</td><td>String</td><td>Yes</td><td>Type of operation. Options are: 'action', 'decision' and 'delay'.</td></tr>
<tr><td>operation_name</td><td>String</td><td>Yes, for 'action' and 'decision' types.</td><td>Name of the operation.</td></tr>
<tr><td>function_name</td><td>String</td><td>Yes, for 'action' and 'decision' types.</td><td>Name of the function.</td></tr>
<tr><td>forwarding_behavior</td><td>String</td><td>No</td><td>Forwarding behavior for the function. One of:<code>always forward</code>: The function always forwards the object to the next workflow.<code>stop this workflow</code>: The function stops this workflow and the object moves to the next workflow.<code>stop all workflows</code>: The function stops all workflows for this object.Default is <code>always forward</code>. Only valid for 'action' and 'decision' types.</td></tr>
<tr><td>function_args</td><td>JSON Object</td><td>No</td><td>Arguments for the function.</td></tr>
<tr><td>duration</td><td>Integer</td><td>Yes, for 'delay' type.</td><td>Length of time before the message goes to the next operation.</td></tr>
<tr><td>reset</td><td>Boolean</td><td>Yes, for 'delay' type.</td><td>Determines whether the timer resets after each occurrence. Not</td></tr>
</table>
</td>
</tr>
</table>

				available if you have set a workflow to **`first_match_only`**. The timer is reset only if an occurrence with the same ID is received (alert_id or situation_id) within the current 'delay' timeframe.

Response

Endpoint **`createWorkflow`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Type	Description
JSON Object	A JSON object containing the ID of the newly created workflow.

Examples

The following examples demonstrate typical use of endpoint **`createWorkflow`**:

Request example

Example cURL request to create a new workflow with a setCustomInfoValue action:

```
curl -X POST -u graze:graze -k \
     -v "https://localhost/graze/v1/createWorkflow" \
     -H "Content-Type: application/json; charset=UTF-8" \
     --data ' {
        "first_match_only": false,
        "operations": [
            {
                "duration": 0,
                "reset": false,
                "type": "delay"
            },
            {
                "operation_name": "set support team value in custom info",
                "function_name": "setCustomInfoValue",
                "forwarding_behavior": "always forward",
                "function_args": {
                    "value": "NOC",
                    "key": "support_team"
                },
                "type": "action"
```

```
            }

        ],
        "moolet_name": "Alert Workflows",
        "workflow_name": "Alert Workflow Example",
        "entry_filter": "state = 9",
        "active": true,
        "description": "Alert Workflow API Example",
        "sweep_up_filter": "((agent = \"Test\") AND (significance = 0)) OR
(severity = 0)"
        }'
```

Response example

Example response returning the new workflow ID:

```
{"id":12}
```

deassignAlert

A Graze API POST request that deassigns the current owner from the specified alert, and leaves it unassigned.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`deassignAlert`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`alert_id`	Number	Yes	ID of the alert that you want to deassign the owner from.

Response

Endpoint **`deassignAlert`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`deassignAlert`**:

Request example

Example cURL request to deassign the current owner from alert ID 7:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/deassignAlert" -H
"Content-Type: application/json; charset=UTF-8" -d '{"alert_id" : 7}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deassignSituation

A Graze API POST request that deassigns the current moderator from the Situation for a specified Situation ID.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deassignSituation** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.

Response

Endpoint **deassignSituation** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **deassignSituation**:

Request example

Example cURL request to deassign the current moderator from Situation 7:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/deassignSituation"
-H "Content-Type: application/json; charset=UTF-8" -d '{"sitn_id" : 7}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deleteCookbook

A Graze API POST request that deletes an existing Cookbook.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deleteCookbook** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the Cookbook that you want to delete.

Response

Endpoint **deleteCookbook** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **deleteCookbook**:

Request example

Example cURL request to delete Cookbook "GrazeCookBook1":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/deleteCookbook" -H
"Content-Type: application/json; charset=UTF-8" -d '{"name" : "GrazeCookBook1"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deleteMaintenanceWindow

A Graze API POST request that deletes a single maintenance window.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deleteMaintenanceWindow** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

id	Number	Yes	ID of the maintenance window you want to delete.

Response

Endpoint **deleteMaintenanceWindow** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **deleteMaintenanceWindow**:

Request example

Example cURL request to delete maintenance window 123:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/deleteMaintenanceWindow" -H "Content-Type:
application/json; charset=UTF-8" -d '{"id"[123]}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deleteMaintenanceWindows

A Graze API POST request that deletes maintenance windows that match the specified filter.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deleteMaintenanceWindows** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
filter	String	Yes	An SQL-like filter to match maintenance windows that you want to delete.

Response

Endpoint **deleteMaintenanceWindows** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **deleteMaintenanceWindows**:

Request examples

Example cURL request to delete maintenance windows that match the filter where the maintenance window ID is 3 or 4:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/deleteMaintenanceWindows" -H "Content-Type:
application/json; charset=UTF-8" -d '{"filter":"id in (3,4)"}'
```

Example cURL request to delete maintenance windows that match the filter where the host is "CSF_RD_243:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/deleteMaintenanceWindows" -H "Content-Type:
application/json; charset=UTF-8" -d '{"filter":"host matches \"CSF_RD_243\""}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deleteMergeGroup

A Graze API POST request that deletes an existing custom merge group.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deleteMergeGroup** takes the following request arguments:

Name	Type	Required	Description
name	String	Yes	Name of the merge group to delete.

Response

Endpoint **deleteMergeGroup** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **deleteMergeGroup**:

Request example

Example cURL request to delete the custom merge group "Merge Group 1":

```
curl -X POST -u graze:graze -k "https://localhost/graze/v1/deleteMergeGroup"
-H "Content-Type: application/json; charset=UTF-8"
--data '{
            "name" : "Merge Group 1"
        }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deleteRecipe

A Graze API POST request that deletes an existing Cookbook Recipe. You can use this endpoint to delete Recipes of all recipe types: Value Recipe, Value Recipe V2, and Bot Recipe.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deleteRecipe** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the Cookbook Recipe that you want to delete.

Response

Endpoint **deleteRecipe** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **deleteRecipe**:

Request example

Example cURL request to delete Recipe "GrazeRecipe1":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/deleteRecipe" -H
"Content-Type: application/json; charset=UTF-8" -d '{"name" : "GrazeRecipe1"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deleteTeam

A Graze API POST request that deletes a single team.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deleteTeam** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	No	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
team_id	String	Yes	ID of the team you want to delete.

Response

Endpoint **deleteTeam** returns the following response:

Type	Description
HTTP	HTTP status or error code indicating request success or failure. See HTTP status code

Code	definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **deleteTeam**:

Request example

Example cURL request to delete team ID 33.

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/deleteTeam" --
data-urlencode 'team_id=33'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deleteTempus

A Graze API POST request that deletes an existing Tempus Moolet.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deleteTempus** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the Tempus Moolet you want to delete.

Response

Endpoint **deleteTempus** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **deleteTempus**:

Request example

Example cURL request to delete Tempus algorithm "newTempus":

```
curl -X POST -u graze:graze -k "https://localhost/graze/v1/deleteTempus" -H
"Content-Type: application/json; charset=UTF-8" --data '{ "name" :
"newTempus" }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

deleteWorkflow

A Graze API POST request that deletes a workflow from the Workflow Engine.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **deleteWorkflow** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
id	Integer	Yes	ID of the workflow you want to delete.

Response

Endpoint **deleteWorkflow** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **deleteWorkflow**:

Request example

Example cURL request to delete workflow ID 12:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/deleteWorkflow" -H
"Content-Type: application/json; charset=UTF-8" --data "{"id":12}"
```

Response example

A successful request returns the HTTP code 200 and no response text.

/enrichment

A Graze API endpoint that allows you to add and delete records from the enrichment data store. To use the Enrichment API, you must install Cisco Add-ons v1.4 or later and set up the Enrichment API.

After you load data into the enrichment data store, you can add the data to an alert's **custom_info** object under the **enrichment** key using the Enrichment Workflow Engine getEnrichment function.

Back to Graze API EndPoint Reference.

POST

The enrichment endpoint only supports the POST HTTP method to create, update, and delete enrichment records.

Request arguments

Endpoint **enrichment** takes the following request payload:

Name	Type	Required	Description
action	String	yes	One of **post** or **delete**. **post** creates or updates enrichment records.

			delete removes enrichment records.
data	Array	yes	An array of of enrichment data records represented as JSON objects.
attribute	String	yes	Name of the alert field or other key for the enrichment data. For example, "source". For the **delete** action, accepts the * wildcard to delete all attributes.
value	String	yes	The value for the associated attribute. For example if the attribute is "source" for host name data, a value might be "sflinux101". For the delete action, accepts the * wildcard at the beginning of the search string,end of the search string or booth to delete all matching values. For example "*linux*" would delete all matching values that contain the string "linux": "SFlinux101", "SFlinux", and "linux101".
enrichment	JSON object	for **post** action	JSON representation of the enrichment data to add to an alert based upon the match attribute and value. For example if you wanted to store store location data: **{"location":"1265 Battery St., San Francisco, CA"}**

Example payload:

```
{"action":"post",
 "data":[
    {"attribute":"source",
    "value":"SFlinux101",
    "enrichment": {
        "location":"1265 Battery St., San Francisco, CA",
        "support_group":"SF NOC"}
    },
    {"attribute":"source",
     "value":"DENlinux102",
        "enrichment": {
            "location":"1700 Lincoln Street, Denver, CO",
            "support_group":"DENVER NOC"}
    }]
}
```

Endpoint **enrichment** takes the following request argument:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

Response

Endpoint **enrichment** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **grazeApiEndpointName**:

Request example

Example cURL request to create enrichment data:

```
curl -k -X POST 'https://localhost/graze/v1/integrations/enrichment' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u graze:graze \
-d '{"action":"post", "data":[ {"attribute":"source", "value":"SFlinux101",
"enrichment": { "location":"1265 Battery St., San Francisco, CA",
"support_group":"SF NOC"} }, {"attribute":"source", "value":"DENlinux102",
"enrichment": { "location":"1700 Lincoln Street, Denver, CO",
"support_group":"DENVER NOC"} }]}'
```

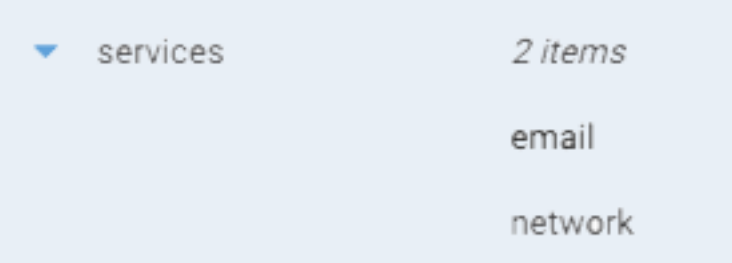
Response example

A successful request returns the HTTP code 200 and no response text.

Match List Items in Recipes

You can create Cookbook Recipes and configure clustering around the use of list-based fields in alert custom info. You can also set whether list-based clustering of a custom field is applied. If not, the field is treated as a string.

A list in custom info is a properly formed JavaScript array. To see if a custom info item is a list, examine the custom info details in the UI. If the list can be expanded and has a value of *x* items at the top level, then it is a list. For example:

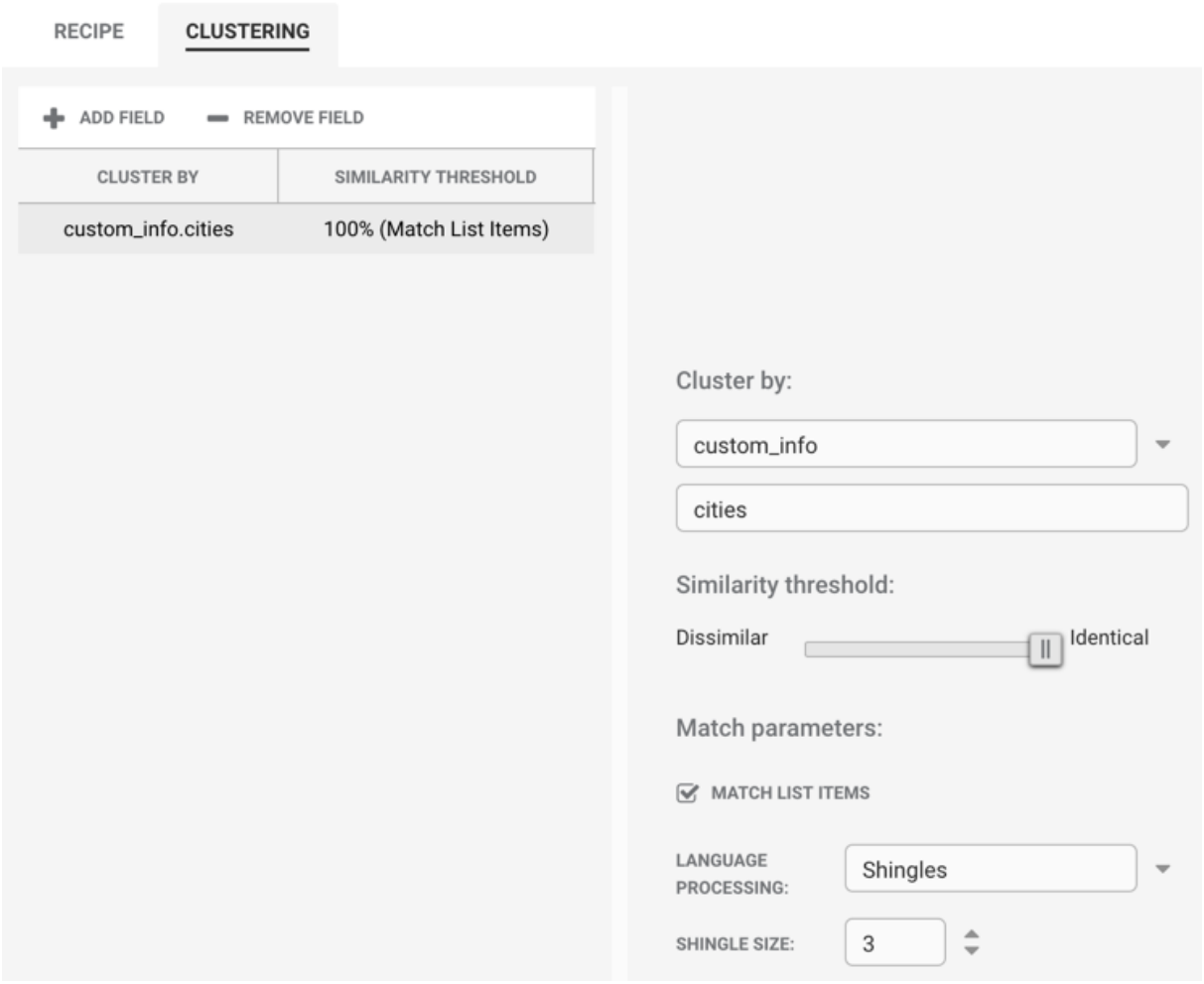


A text field containing comma separated values is not considered a list.

Configure Match List Items for a Custom Info Field

To match list items for a custom_info field:

- On the Settings tab, select Cookbook Recipes from the Algorithms section, select the Recipe you want to configure, and click on the Clustering tab.
- In the Cluster By field, select the custom_info attribute from the drop-down list. Enter the custom_info field name in the box below.
- Check the Match List Items check box to match individual items in custom_info lists and use the slider to select the similarity threshold for this custom_info field.



Comparison of Match List Items

The Cookbook Recipe applies the similarity threshold that you set to compare each individual item in the list, not all the items in the list.

For example, you have the following lists in two alerts and the similarity threshold is 100%:

```
Alert 1: [ ABC , DEF ]
Alert 2: [ ABC123, DEF123, ABC, DEF ]
```

This results in similarity comparisons between:

- ABC and ABC123
- ABC and DEF123
- ABC and ABC
- ABC and DEF
- DEF and ABC123
- DEF and DEF123
- DEF and ABC
- DEF and DEF

Since there are two identical matches, [ABC and ABC] and [DEF and DEF], the Cookbook Recipe clusters these alerts together.

If you want to calculate the total similarity of list items, that is, how many items in list 1 appear in list 2, you should not select Match List Items and set Language Processor to Words so that the Cookbook Recipe treats the list as a string. In the above example, there is a 50% match of items in both lists, [ABC and DEF], so if the similarity threshold is 100%, the Cookbook Recipe does not cluster these alerts together.

Example

You configure your Recipe to treat the custom_info field 'cities' as a list and set the similarity threshold to 100%, as shown above.

After configuring the Recipe, Cisco Crosswork Situation Manager receives the following four alerts:

```
Alert 1: custom_info.cities = ["London"]
Alert 2: custom_info.cities = ["London", "San Francisco", "Venice", "Bangalore"]
Alert 3: custom_info.cities = ["Venice", "Bangalore"]
Alert 4: custom_info.cities = ["Bangalore"]
```

This configuration would produce four candidate clusters:

- Cluster A: Alert 1 and alert 2 match on `"London"`.
- Cluster B: Alert 2 matches on `"San Francisco"`.
- Cluster C: Alert 2 and alert 3 match on `"Venice"`.
- Cluster D: Alerts 2, 3 and 4 match on `"Bangalore"`.

Cookbook creates two Situations because cluster D contains all the alerts in clusters B and C:

- Cluster A (alerts 1 and 2) becomes Situation X.
- Clusters B, C, and D (alerts 2, 3, and 4) become Situation Y.

You must be careful when setting the similarity threshold if you are using list-based clustering. If the similarity threshold is low enough, you may end up with Situations containing blended list similarity. In the above example, alert 2 is common to both Situation X (London) and Situation Y (Bangalore). If the similarity were set to 25%, these two Situations would merge.

If the Recipe does not see 'custom_info.cities' field as a list, it treats the field as a single string. This means that, in this example, all four alerts would end up in separate Situations with no clustering.

getActiveSituationIds

A Graze API GET request that returns the total number of active Situations, and a list of their Situation IDs. Active Situations are those that are not Closed, Resolved or Dormant.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getActiveSituationIds`** takes the following request argument:

Name	Type	Required	Description

auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

Endpoint **getActiveSituationIds** takes no other arguments because this endpoint returns data on all active Situations.

Response

Endpoint **getActiveSituationIds** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
total_situations	Number	Total number of active Situations.
sitn_ids	Array	A list of active Situation IDs.

Examples

The following examples demonstrate typical use of endpoint **getActiveSituationIds**:

Request example

Example cURL request to return all active Situations in Cisco Crosswork Situation Manager:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getActiveSituationIds"
```

Response example

Example response returning the IDs of ten Situations:

```
{
   "total_situations": 10,
   "sitn_ids": [4, 5, 6, 12, 14, 15, 16, 17, 18, 19]
}
```

getAlertActions

A Graze API GET request that returns the actions for one or more alerts, ordered most recent last. You can use the **from** and **to** arguments to specify a period that you want to retrieve alert actions for. If you do not specify these, actions for all dates and times are returned.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getAlertActions** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
alert_ids	Array of Numbers	No	List of alert IDs.
start	Integer	Yes	Starting row from which data should be included.

limit	Integer	Yes	Maximum number of actions you want to return.
actions	Array of Numbers	No	List of action codes. If no action codes are specified, all action codes are returned. See Alert Action Codes for a list of action codes and their descriptions. Only action codes 8 (Alert Resolved) and 9 (Alert Closed) are valid.
from	Number	No	Start time of the period you want to retrieve alert actions for. This is in Unix epoch time in seconds.
to	Number	No	End time of the period you want to retrieve alert actions for. This is in Unix epoch time in seconds.

Response

Endpoint **getAlertActions** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object which contains alert details from the following:

Name	Type	Description
uid	Number	User ID.
action_code	Number list	Code for the action in the JSON object. See Alert Action Codes for a list of action codes and their descriptions.
description	String	Description of the action.
details	String	Details of the action.
type	String	Type of action.
alert_id	Integer	Alert ID.
timed_at	Integer	Timestamp of the action.

Examples

The following examples demonstrate typical use of endpoint **getAlertActions**:

Request Examples

Example cURL request to return the first 50 actions for alert IDs 1, 2, 3, and 6 for action codes 8 (Alert Resolved) and 9 (Alert Closed):

```
curl -G -u graze:graze -k -v
"https://docsdev.moog.cloud/graze/v1/getAlertActions" --data-urlencode
'alert_ids=[1, 2, 3, 6]' --data-urlencode 'actions=[8, 9]' --data-urlencode
'limit=50' --data-urlencode 'start=0'curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getAlertActions" --data-urlencode 'alert_ids=[1, 2,
3, 6]' --data-urlencode 'actions=[8, 9]' --data-urlencode 'limit=50' --data-
urlencode 'start=0'
```

Example cURL request to return the first 50 actions for action codes 8 (Alert Resolved) and 9 (Alert Closed) between Unix epoch times 1553861746 and 1553872546:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getAlertActions" --
data-urlencode 'actions=[8, 9]' --data-urlencode 'limit=50' --data-urlencode
'start=0' --data-urlencode 'from=1553861746' --data-urlencode 'to=1553872546'
```

Response example

Example response returning the actions for alert ID 313:

```
[{
      "uid": 49,
      "action_code": 9,
      "description": "Alert Closed",
      "details": {},
      "alert_id": 313,
      "timed_at": 1557504912
 },{
      "uid": 49,
      "action_code": 8,
      "description": "Alert Resolved",
      "details": {},
      "alert_id": 313,
      "timed_at": 1557504393
 },{
      "uid": 3,
      "action_code": 10,
      "description": "Ran Tool",
      "details": {
              "tool_id": 271,
          "tool": "get data"
          },
      "alert_id": 313,
      "type": "event",
      "timed_at": 1557321088,
      "username": "admin"
}]
```

getAlertDetails

A Graze API GET request that returns details, such as the description or severity, of an alert.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getAlertDetails`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`alert_id`	Number	Yes	Alert ID.

Response

Endpoint **`getAlertDetails`** returns the following response:

Type	Description
HTTP	HTTP status or error code indicating request success or failure. See HTTP status code

Code	definitions for more information.

Successful requests return a JSON object which contains alert details from the following:

Name	Type	Description
active_sitn_list	Number list	A list of Situation IDs of the active Situations to which this alert belongs.
agent	String	Agent name associated with this alert. *
agent_location	String	Agent location associated with this alert. *
alert_id	Number	Alert ID.
class	String	Class associated with this alert. *
count	Number	Number of times that this alert has occurred.
custom_info	JSON object	A JSON object containing the custom information.
description	String	Description associated with this alert. *
entropy	Number	Entropy value of the alert, the measure of probability that an alert will arrive in the system at any given time. This is a value between 0 (very certain) and 1 (very uncertain).
external_id	String	External ID associated with this alert. *
first_event_time	Number	Timestamp (in Unix epoch time) of the first occurrence of the alert.
int_last_event_time	Number	Internal Cisco Crosswork Situation Manager timestamp (in Unix epoch time) of the last occurrence of this alert.
last_event_time	Number	Timestamp (in Unix epoch time) of the last occurrence of this alert.
last_state_change	Number	Timestamp (in Unix epoch time) of the last state change of this alert.
manager	String	Manager name associated with this alert. *
owner	Number	ID of the user that this alert is assigned to.
severity	Number	The severity of the alert as an integer:0 = Clear1 = Indeterminate2 = Warning3 = Minor4 = Major5 = Critical
signature	String	Unique alert identifier.
significance	Number	Significance of the alert as an integer: 0 = Collateral 1 = Related 2 = Impacting 3 = Causal

source	String	Source associated with this alert. *
source_id	String	Source ID associated with the alert. *
state	Number	Indicates the lifecycle state of the alert.
type	String	Type associated with this alert. *

* = These details are derived from the input event text field, via the LAMs.

Examples

The following examples demonstrate typical use of endpoint **getAlertDetails**:

Request example

Example cURL request to return the details for alert ID 3968:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getAlertDetails" --
data-urlencode "alert_id=3968"
```

Response example

Example response returning the details of alert ID 3968:


```
{
   "active_sitn_list":[1],
   "agent":"TestBed",
   "agent_location":"localhost",
   "alert_id":3968,
   "class":"WebMon",
   "count":2,
   "custom_info":null,
   "description":"Web Server HTTPD is DOWN",
   "external_id":"12345",
   "first_event_time":1416307126,
   "int_last_event_time":1416307188,
   "last_event_time":1416307131,
   "last_state_change":1416307144,
   "manager":"WebMon",
   "owner":2,
   "severity":0,
   "signature":"SIG:Web Server Down Trap:xldn1458pap:10",
   "significance":3,
   "source":"xldn1458pap",
   "source_id":"xldn1458pap",
   "state":9,
   "type":"HTTPDDown"
}
```


getAlertIds

A Graze API GET request that returns the total number of alerts, and a list of the alert IDs, for a specified alert filter and a limit.

Note

Take special care when using endpoint **getAlertIds**. Overuse of this endpoint can have a negative impact on the backend datastore.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getAlertIds** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
query	String	Yes	An SQL-like filter that alerts must match to be returned.
limit	Number	Yes	Maximum number of alert IDs to return.

Response

Endpoint **getAlertIds** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object which contains alert details from the following:

Name	Type	Description
total_alerts	Number	Total number of alerts, or unique alerts.
alert_ids	JSON Array	A list of alert IDs.

Examples

The following examples demonstrate typical use of endpoint **getAlertIds**:

Request example

Example cURL request to return the first 20 alert IDs that satisfy the filter where the agent is not SYSLOG and the description matches "AUTH-SERVICE":

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getAlertIds" --data-
urlencode 'query=agent!=SYSLOG and description matches "AUTH-SERVICE"' --data-
urlencode 'limit=20'
```

Response example

Example response returning the first 20 alert IDs:

```
{
    "total_alerts":20,

"alert_ids":[78,234,737,1253,1459,1733,2166,2653,2855,3133,3414,3538,3729,3905,3
991,4110,4160,4536,4692,4701]
}
```

getAllSessionInfo

A Graze API GET request that returns session information for all users over a period of time.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getAllSessionInfo`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	No	Start time of the period you want to retrieve session information for. This is in Unix epoch time in seconds. If empty, returns all session information for all users.
`to`	Number	No	End time of the period you want to retrieve session information for. This is in Unix epoch time in seconds. If empty, returns user records to date.
`start`	Number	No	Starting record from which data should be included. Default is 0, the first record.
`limit`	Number	No	Maximum number of records you want to return. Default is 200.

Response

Endpoint **`getAllSessionInfo`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`sessionId`	Number	ID of the session.
`userName`	String	User name for the session.
`startTime`	Number	Start time of the session, in Unix epoch time.
`lastAccess`	Number	Last access time within the session, in Unix epoch time.

Examples

The following examples demonstrate typical use of endpoint **`getAllSessionInfo`**:

Request example

Example cURL request to return session information for all users:

```
curl -G -u graze:graze -k "https://localhost/graze/v1/getAllSessionInfo" --data-
urlencode "from=1578655174" --data-urlencode "start=1" --data-urlencode
"limit=6"
```

Response example

Example response returning six session information records for all users, starting at record 2:

```
[
    {
        "sessionId": 2,
        "userName": "user1",
        "startTime": 1571666244,
        "lastAccess": 1571666301
    },
```

```
        {
            "sessionId": 3,
            "userName": "admin",
            "startTime": 1571666307,
            "lastAccess": 1571666760
        },
        {
            "sessionId": 4,
            "userName": "user3",
            "startTime": 1571666764,
            "lastAccess": 1571673384
        },
        {
            "sessionId": 5,
            "userName": "user1",
            "startTime": 1571735292,
            "lastAccess": 1571738917
        },
        {
            "sessionId": 6,
            "userName": "graze",
            "startTime": 1571784397,
            "lastAccess": 1571784401
        },
        {
            "sessionId": 7,
            "userName": "admin",
            "startTime": 1571799980,
            "lastAccess": 1571800581
        }
    ]
```


getCookbooks

A Graze API GET request that returns all the Cookbooks in Cisco Crosswork Situation Manager.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getCookbooks`** takes the following request argument:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

Endpoint **`getCookbooks`** takes no other arguments because this endpoint returns data on all Cookbooks in Cisco Crosswork Situation Manager.

Response

Endpoint **`getCookbooks`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Type	Description
List of JSON Objects	A list of all the Cookbooks in Cisco Crosswork Situation Manager and all their details.

Examples

The following examples demonstrate typical use of endpoint **`getCookbooks`**:

Request example

Example cURL request to return all the Cookbooks in Cisco Crosswork Situation Manager:

```
curl -X GET -u graze:graze -k -v "https://localhost/graze/v1/getCookbooks"
```

Response example

Example response returning the details of all Cookbooks in Cisco Crosswork Situation Manager:


```
[
    {
        "name": "Default Cookbook",
        "classname": "CCookbook",
        "metric_path_moolet": true,
        "description": null,
        "run_on_startup": true,
        "process_output_of": [
            "MaintenanceWindowManager"
        ],
        "id": 1,
        "scale_by_severity": false,
        "entropy_threshold": 0.0,
        "first_recipe_match_only": true,
        "cluster_by": "first_match",
        "cook_for": 900,
        "cook_for_extension": 0,
        "max_cook_for": null,
        "moobot": "Cookbook.js",
        "recipes": [
            "Criticals",
            "Description",
            "Source"
        ],
        "threshold_type": "global"
    },
    {
        "name": "BotCookBook1",
        "classname": "CCookbook",
        "metric_path_moolet": true,
        "description": null,
        "run_on_startup": true,
        "process_output_of": [
            "Alert Workflows"
        ],
        "id": 2,
        "scale_by_severity": false,
        "entropy_threshold": 0.25,
        "first_recipe_match_only": false,
        "cluster_by": "first_match",
        "cook_for": 3600,
        "cook_for_extension": 0,
```

```
            "max_cook_for": 0,
            "moobot": "Cookbook.js",
            "recipes": [
                "BotRecipe1",
                "BotRecipe2"
            ],
            "threshold_type": "explicit_value"
        },
        {
            "name": "GrazeCookbook3",
            "classname": "CCookbook",
            "metric_path_moolet": true,
            "description": null,
            "run_on_startup": true,
            "process_output_of": [
                "Alert Workflows"
            ],
            "id": 3,
            "scale_by_severity": false,
            "entropy_threshold": 0.22,
            "first_recipe_match_only": false,
            "cluster_by": "first_match",
            "cook_for": 3600,
            "cook_for_extension": 0,
            "max_cook_for": 0,
            "moobot": "Cookbook.js",
            "recipes": [
                "GrazeRecipe1",
                "GrazeRecipe2",
                "GrazeRecipe3"
            ],
            "threshold_type": "explicit_value"
        }
    ]
```


getDefaultMergeGroup

A Graze API GET request that returns details of the default merge group in Cisco Crosswork Situation Manager.

Clustering algorithms, such as Cookbook and Tempus, use the default values in the default merge group unless you have set up custom merge groups with different values to merge Situations from these clustering algorithms. You can set up merge groups using the UI (see Merge Groups for details) or using the Graze API endpoint addMergeGroup.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getDefaultMergeGroup`** takes the following request argument:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

Endpoint **`getDefaultMergeGroup`** takes no other arguments because this endpoint returns details of the default merge group in Cisco Crosswork Situation Manager.

Response

Endpoint **getDefaultMergeGroup** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
alert_threshold	Integer	Minimum number of alerts that must be present in a cluster before it can become a Situation in the merge group. Default value is 1.
situation_similarity_limit	Floating Point	Percentage of alerts two Situations must share before they are merged for this group. A value between 0 and 1. Default value is 0.7.

Examples

The following examples demonstrate typical use of endpoint **getDefaultMergeGroup**:

Request example

Example cURL request to return the default merge group in Cisco Crosswork Situation Manager:

```
curl -G
-u graze:graze
-k "https://example.com/graze/v1/getDefaultMergeGroup"
```

Response example

Example response returning details of the default merge group in Cisco Crosswork Situation Manager:

```
{
    "alert_threshold": 1,
    "situation_similarity_limit": 0.7
}
```

getEventsAnalyserConfig

A Graze API GET request that returns the details of the Events Analyser configuration, including the priority words and stop words.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getEventsAnalyserConfig** takes the following request argument:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

Endpoint **getEventsAnalyserConfig** takes no other arguments.

Response

Endpoint **getEventsAnalyserConfig** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
priority_words		

Examples

The following examples demonstrate typical use of endpoint **getEventsAnalyserConfig**:

Request example

Example cURL request to return the details of the Events Analyser configuration:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getEventsAnalyserConfig"
```

Response examples

Example response returning an Events Analyser configuration that does not use partitioning:


```
{
    "priority_words": false,
    "partition_by": "",
    "stop_word_length": 0,
    "stemming_language": "english",
    "partition_overrides": null,
    "priority_words_list": [],
    "stop_words": true,
    "stop_words_list": [
        "%",
        ":",
        "=",
        ".",
        "|",
        "-",
        "~",
        "&",
        "a",
        "able",
        "about",
        "across",
        "after",
        "all",
        "almost",
        "also",
        "am",
        "among",
        "an",
        "and",
        "any",
        "are",
```

```
"as",
"at",
"be",
"because",
"been",
"but",
"by",
"can",
"cannot",
"could",
"dear",
"did",
"do",
"does",
"either",
"else",
"ever",
"every",
"for",
"from",
"get",
"got",
"had",
"has",
"have",
"he",
"her",
"hers",
"him",
"his",
"how",
"however",
"i",
"if",
"in",
"into",
"is",
"it",
"its",
"just",
"least",
"let",
"like",
"likely",
"may",
"me",
"might",
"most",
"must",
"my",
"neither",
"no",
"nor",
"not",
"of",
"off",
"often",
"on",
"only",
"or",
"other",
```

```
        "our",
        "own",
        "rather",
        "said",
        "say",
        "says",
       "she",
        "should",
        "since",
        "so",
        "some",
        "than",
        "that",
        "the",
        "their",
        "them",
        "then",
        "there",
        "these",
        "they",
        "this",
        "tis",
        "to",
        "too",
        "twas",
        "us",
        "wants",
        "was",
        "we",
        "were",
        "what",
        "when",
        "where",
        "which",
        "while",
        "who",
        "whom",
        "why",
        "will",
        "with",
        "would",
        "yet",
        "you",
        "your"
    ],
    "stemming": false,
    "id": 1,
    "casefold": true,
    "fields": [
        "description"
    ],
    "mask": {
        "path": false,
        "number": true,
        "date_time": true,
        "mac_address": false,
        "guid": false,
```

```
        "hex": false,
        "oid": false,
        "ip_address": false,
        "stop_word": false,
        "word": false,
        "url": false,
        "email": false
    }
}
```


Example response returning an Events Analyser configuration that uses partitioning:


```
{
    "priority_words": true,
    "partition_by": "source",
    "stop_word_length": 0,
    "stemming_language": "english",
    "partition_overrides": {
        "SOURCE11": {
            "stop_words": false,
            "priority_words": false,
            "fields": [
                "description"
            ],
            "casefold": true,
            "stop_word_length": 5
        },
        "SOURCE22": {
            "priority_words_list": [
                "reboot",
                "shutdown"
            ],
            "stop_words": true,
            "stop_words_list": [
                "france",
                "germany",
                "italy",
                "peru",
                "india",
                "japan",
                "korea"
            ],
            "stemming": true,
            "priority_words": true,
            "stop_word_length": 1,
            "mask": {
                "date_time": false,
                "ip_address": true
            }
        }
    },
    "priority_words_list": [
        "down",
        "fail",
        "loss",
        "low"
    ],
    "stop_words": true,
    "stop_words_list": [
        "%",
        ":",
        "=",
```

```
"."
"|"
"-"
"~"
"&"
"a"
"able"
"about"
"across"
"after"
"all"
"almost"
"also"
"am"
"among"
"an"
"and"
"any"
"are"
"as"
"at"
"be"
"because"
"been"
"but"
"by"
"can"
"cannot"
"could"
"dear"
"did"
"do"
"does"
"either"
"else"
"ever"
"every"
"for"
"from"
"get"
"got"
"had"
"has"
"have"
"he"
"her"
"hers"
"him"
"his"
"how"
"however"
"i"
"if"
"in"
"into"
"is"
"it"
```

```
"its",
"just",
"least",
"let",
"like",
"likely",
"may",
"maybe",
"me",
"might",
"most",
"must",
"my",
"neither",
"no",
"nor",
"not",
"of",
"off",
"often",
"on",
"only",
"or",
"other",
"our",
"own",
"rather",
"said",
"say",
"says",
"she",
"should",
"since",
"so",
"some",
"than",
"that",
"the",
"their",
"them",
"then",
"there",
"these",
"they",
"this",
"tis",
"to",
"too",
"twas",
"us",
"wants",
"was",
"we",
"were",
"what",
"when",
"where",
"which",
"while",
"who",
"whom",
```

```
            "why",
            "will",
            "with",
            "would",
            "yet",
            "you",
            "your"
        ],
        "stemming": false,
        "id": 1,
        "casefold": true,
        "fields": [
            "description"
        ],
        "mask": {
            "path": false,
            "number": true,
            "date_time": true,
            "mac_address": false,
            "guid": false,
            "hex": false,
            "oid": false,
            "ip_address": false,
            "stop_word": false,
            "word": false,
            "url": false,
            "email": false
        }
    }
```

getEventsAnalyserPartitionOverrides

A Graze API GET request that returns the partition override details in the Events Analyser configuration.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getEventsAnalyserPartitionOverrides`** takes the following request argument:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

Endpoint **`getEventsAnalyserPartitionOverrides`** takes no other arguments because this endpoint returns data on all the partition overrides in the Events Analyser configuration.

Response

Endpoint **`getEventsAnalyserPartitionOverrides`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Type	Description
JSON Object	Details of the partition overrides in the Events Analyser.

Examples

The following examples demonstrate typical use of endpoint **`getEventsAnalyserPartitionOverrides`**:

Request example

Example cURL request to return the partition overrides in the Events Analyser configuration:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getEventsAnalyserPartitionOverrides"
```

Response example

Example response returning the partition overrides for the Events Analyser:

```
{
    "SOURCE11": {
        "stop_words": false,
        "priority_words": false,
        "fields": [
            "description"
        ],
        "casefold": true,
        "stop_word_length": 5
    },
    "SOURCE22": {
        "priority_words_list": [
            "reboot",
            "shutdown"
        ],
        "stop_words": true,
        "stop_words_list": [
            "france",
            "germany",
            "italy",
            "peru",
            "india",
            "japan",
            "korea"
        ],
        "stemming": true,
        "priority_words": true,
        "stop_word_length": 1,
        "mask": {
            "date_time": false,
            "ip_address": true
        }
    }
}
```

getEventsAnalyserWords

A Graze API GET request that returns the list of priority words or stop words used by the Events Analyser. This endpoint returns the stop words or priority words for the default partition, depending on the argument you supply.

See getEventsAnalyserConfig to return the main Events Analyser configuration.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getEventsAnalyserWords** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
type	String	Yes	Determines whether the endpoint returns a list of stop words or priority words. Set to **priority_word** for the list of priority words. Set to **stop_word** for the list of stop words.

Response

Endpoint **getEventsAnalyserWords** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return the following:

Type	Description
JSON Array	List of priority words or stop words, depending on the request argument **type**.

Examples

The following examples demonstrate typical use of endpoint **getEventsAnalyserWords**:

Priority words example

Request example

Example cURL request to return the list of priority words:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getEventsAnalyserWords"
--data-urlencode 'type=priority_word'
```

Stop words example

Request example

Example cURL request to return the list of stop words:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getEventsAnalyserWords"
\
--data-urlencode 'type=stop_word'
```

Response example

Example response returning the list of stop words:

```
[
    "%",
    ":",
    "=",
```

```
"."
"|"
"-"
"~"
"&"
"a"
"able"
"about"
"across"
"after"
"all"
"almost"
"also"
"am"
"among"
"an"
"and"
"any"
"are"
"as"
"at"
"be"
"because"
"been"
"but"
"by"
"can"
"cannot"
"could"
"dear"
"did"
"do"
"does"
"either"
"else"
"ever"
"every"
"for"
"from"
"get"
"got"
"had"
"has"
"have"
"he"
"her"
"hers"
"him"
"his"
"how"
"however"
"i"
"if"
"in"
"into"
"is"
"it"
"its"
"just"
"least"
"let"
```

```
"like",
"likely",
"may",
"me",
"might",
"most",
"must",
"my",
"neither",
"no",
"nor",
"not",
"of",
"off",
"often",
"on",
"only",
"or",
"other",
"our",
"own",
"rather",
"said",
"say",
"says",
"she",
"should",
"since",
"so",
"some",
"than",
"that",
"the",
"their",
"them",
"then",
"there",
"these",
"they",
"this",
"tis",
"to",
"too",
"twas",
"us",
"wants",
"was",
"we",
"were",
"what",
"when",
"where",
"which",
"while",
"who",
"whom",
"why",
```

```
        "will",
        "with",
        "would",
        "yet",
        "you",
        "your"
    ]
```

getGlobalEntropyThresholds

A Graze API GET request that returns the global default entropy threshold and all manager-specific entropy thresholds that have been set.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getGlobalEntropyThresholds`** takes the following request argument:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

Endpoint **`getGlobalEntropyThresholds`** takes no other arguments because this endpoint returns data on all entropy thresholds.

Response

Endpoint **`getGlobalEntropyThresholds`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`filter`	JSON Object	Filter for a manager-specific entropy threshold; empty for the global default entropy threshold.
`name`	String	Name of the manager or **`default`** for the global default entropy threshold.

Examples

The following examples demonstrate typical use of endpoint **`getGlobalEntropyThresholds`**:

Request example

Example cURL request to return the global default entropy threshold and any manager-specific entropy thresholds:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getGlobalEntropyThreshold"
```

Response example

Example response returning the global default entropy threshold, and two manager entropy thresholds for `"rough-river"` and `"lingering-mountain"`:

```
[
    {
        "filter": "{\"op\": 0, \"type\": \"LEAF\", \"value\": \"rough-
river\", \"column\": \"manager\"}",
        "name": "rough-river",
        "type": "percentage_reduction",
        "value": 0.327
    },
    {
        "filter": "{\"op\": 0, \"type\": \"LEAF\", \"value\": \"lingering-
mountain\", \"column\": \"manager\"}",
        "name": "lingering-mountain",
        "type": "entropy_value",
        "value": 0.5002021249999999
    },
    {
        "filter": "{}",
        "name": "default",
        "type": "percentage_reduction",
        "value": 0.19065887902499268
    }
]
```


getIntegrationConfig

A Graze API GET request that exports the configuration and mapping needed for an integration in JSON format.

The exported JSON file can be saved as a duplicate configuration of the original integration. For example, you can modify and save the returned object as **`webhook_lam_custom.conf`**. Run it with this command:

```
$MOOGSOFT_HOME/bin/webhook_lam --config=webhook_lam_custom.conf
```

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getIntegrationConfig`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`integration_id`	Number	Yes	A unique identifier given to each integration by Cisco Crosswork Situation Manager.

Response

Endpoint **`getIntegrationConfig`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
config	Object	An object containing the integration configuration details.

Examples

The following examples demonstrate typical use of endpoint **getIntegrationConfig**:

Request example

Example cURL request to return the configuration for the integration with ID 1:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getIntegrationConfig?integration_id=1"
```

Response example

Example return of a JSON object containing the integration configuration details:

```
{
    "config": {
        "filter": {
            "presend": "WebhookLam.js"
        },
        "process": "webhook_lam_webhook1",
        "conversions": {
            "sevConverter": {
                "output": "INTEGER",
                "lookup": "severity",
                "input": "STRING"
            },
            "stringToInt": {
                "output": "INTEGER",
                "input": "STRING"
            }
        },
        "mapping": {
            "catchAll": "overflow",
            "rules": [
                {
                    "name": "signature",
                    "rule": "$signature"
                },
                {
                    "name": "source_id",
                    "rule": "$source_id"
                },
                {
                    "name": "external_id",
                    "rule": "$external_id"
                },
                {
                    "name": "manager",
                    "rule": "$manager"
                },
                {
                    "name": "source",
                    "rule": "$source"
                },
                {
                    "name": "class",
                    "rule": "$class"
                },
```

```
            {
                "name": "agent",
                "rule": "$LamInstanceName"
            },
            {
                "name": "agent_location",
                "rule": "$agent_location"
            },
            {
                "name": "type",
                "rule": "$type"
            },
            {
                "name": "severity",
                "rule": "$severity",
                "conversion": "sevConverter"
            },
            {
                "name": "description",
                "rule": "$description"
            },
            {
                "name": "agent_time",
                "rule": "$agent_time",
                "conversion": "stringToInt"
            }
        ]
    },
    "agent": {
        "name": "webhook_lam_webhook1"
    },
    "monitor": {
        "address": "localhost",
        "authentication_cache": true,
        "use_ssl": false,
        "auto_port_assign": true,
        "authentication_type": "basic",
        "rpc_response_timeout": 20,
        "lists_contain_multiple_events": true,
        "proxy": "https://freida7/events/webhook_webhook1",
        "accept_all_json": true,
        "port": 51000,
        "name": "Webhook Lam Monitor (Webhook1)",
        "num_threads": 5,
        "rest_response_mode": "on_receipt",
        "class": "CRestMonitor"
    },
    "constants": {
        "severity": {
            "0": 2,
            "moog_lookup_default": 1,
            "3": 5,
            "5": 4,
            "CLEAR": 0,
            "2": 3,
            "MAJOR": 4,
            "CRITICAL": 5,
```

```
                "MINOR": 3,
                "INDETERMINATE": 1,
                "1": 2
            }
        }
    }
}
```

getMaintenanceWindows

A Graze API GET request that returns maintenance windows based on how many should be returned. Only returns active recurring windows and scheduled maintenance windows, not expired or deleted maintenance windows.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getMaintenanceWindows`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`start`	Number	Yes	Number of the first maintenance window to return, 0 to start at the first, 10 to start at the 11th.
`limit`	Number	Yes	Maximum number of maintenance windows to return.

Response

Endpoint **`getMaintenanceWindows`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`windows`	Array	An array of objects containing the details of the returned maintenance windows.

Examples

The following examples demonstrate typical use of endpoint **`getMaintenanceWindows`**:

Request example

Example cURL request to return the first 20 maintenance windows:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getMaintenanceWindows"
--data-urlencode 'start=0' --data-urlencode 'limit=20'
```

Response example

Example successful response returning details of one maintenance window:

```
{
    "windows": [
        {
            "del_flag": false,
            "forward_alerts": false,
```

```
            "last_updated": 1574074779,
            "timezone": "Europe/London",
            "description": "This is new maintenance window",
            "recurring_period_units": 0,
            "filter": "(severity IN (0, 1, 2, 3, 4, 5)) AND (owner IN (3))",
            "duration": 3600,
            "recurring_period": 0,
            "name": "New Maintenance Window",
            "updated_by": 3,
            "id": 2,
            "start_date_time": 1574074744
        }
    ]
}
```


getMergeGroups

A Graze API GET request that returns details of all the custom merge groups in Cisco Crosswork Situation Manager.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getMergeGroups** takes the following argument:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

Endpoint **getMergeGroups** takes no other arguments because this endpoint returns details of all the custom merge group in Cisco Crosswork Situation Manager.

Response

Endpoint **getMergeGroups** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	The merge group's name.
moolets	Array of Strings	List of clustering algorithm Moolets to include in the custom merge group.
alert_threshold	Integer	Minimum number of alerts that must be present in a cluster before it can become a Situation. Must be greater than or equal to 1. Enter **null** if you want the custom merge group to use the default merge group value. Default merge group value is 2.
situation_similarity_limit	Floating	Percentage of alerts two Situations must share before they

	Point	are merged for this group. Enter a value between 0 and 1. Entering **`null`** causes the merge group to use the same value as the default merge group.

Examples

The following examples demonstrate typical use of endpoint **`getMergeGroups`**:

Request example

Example cURL request to return all the custom merge groups in Cisco Crosswork Situation Manager:

```
curl -G
-u graze:graze
-k "https://example.com/graze/v1/getMergeGroups"
```

Response example

Example response returning details of all the custom merge groups in Cisco Crosswork Situation Manager:

```
[
{
    "name":"Default Cookbook",
    "moolets":
    [
        "Default Cookbook"
    ],
    "alert_threshold":2,
    "situation_similarity_limit":0.6
},
{
    "name":"Merge Group 1",
    "moolets":
    [
        "Recipe 1"
        "Recipe 2"
    ],
    "alert_threshold":null,
    "situation_similarity_limit":0.5
},
{
    "name":"Merge Group 2",
    "moolets":
    [
        "Recipe 2"
        "Time Based (Tempus)"
    ],
    "alert_threshold":2,
    "situation_similarity_limit":null
}
]
```

getPrcLabels

A Graze API GET request that returns probable root cause (PRC) information for all alerts or specified alerts within a Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getPrcLabels** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Integer	Yes	Situation ID.
alert_ids	Array of Numbers	No	List of alert IDs.

Response

Endpoint **getPrcLabels** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **getPrcLabels**:

Request example

Example cURL request to return the PRC labels for alerts 1, 2, 3, and 4 in Situation 157:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getPrcLabels?sitn_id=157&alert_ids=[1,2,3,4]"
```

Response example

Example response returning the PRC labels for alerts 1, 2, 3, and 4 in the Situation:

```
{
   "non_causal": [2,3],
   "unlabelled": [4],
   "causal": [1]
}
```

getProcesses

A Graze API GET request that returns a list of the processes in the database.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getProcesses** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
limit	Integer	No	Maximum number of processes to return. Defaults to 100.
exact_match	Boolean	No	If **true**, the query performs an exact match on the process name. If

			false, the query checks for **contains** only on the process name. Defaults to **false**.

Response

Endpoint **getProcesses** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
process_id	Number	ID of the process.
name	String	Name of the process.
description	String	Description of the process.

Examples

The following examples demonstrate typical use of endpoint **getProcesses**:

Request example

Example cURL request to return the first 100 processes containing "Network" in the process name:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getProcesses" --data-
urlencode 'query=Network' --data-urlencode 'exact_match=false'
```

Response example

Example response returning three process names containing "Network":

```
[
    {
        "process_id": 1,
        "name": "Network LON",
        "description": "Network London"
    },
    {
        "process_id": 2,
        "name": "NY Network A",
        "description": "Network New York A"
    },
    {
        "process_id": 3,
        "name": "NY Network B",
        "description": "Network New York B"
    }
]
```

getRecipes

A Graze API GET request that returns all the Recipes in Cisco Crosswork Situation Manager.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getRecipes** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
cookbook	String	No	Name of the Cookbook that you want to return the Recipes for. Do not specify to return all Recipes.

Response

Endpoint **getRecipes** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Type	Description
List of JSON Objects	A list of all the Recipes in Cisco Crosswork Situation Manager and all their details.

Examples

The following examples demonstrate typical use of endpoint **getRecipes**:

Request example

Example cURL request to return all the Recipes in Cisco Crosswork Situation Manager:

```
curl -X GET -u graze:graze -k -v "https://localhost/graze/v1/getRecipes"
```

Response example

Example response returning all the Recipes in Cisco Crosswork Situation Manager:


```
[
    {
        "cookbooks": [
            "Default Cookbook"
        ],
        "name": "Description",
        "id": 1,
        "description": "Alerts with a similar description",
        "alert_threshold": 2,
        "trigger": "severity >= 3",
        "exclusion": "",
        "seed_alert": "",
        "rate": 0,
        "min_sample_size": 5,
        "max_sample_size": 10,
        "cook_for": null,
        "cook_for_extension": 0,
        "max_cook_for": null,
        "cluster_by": "first_match",
        "active": true,
        "chef": "CValueRecipeV2",
        "use_dynamic_topology": false,
        "topology_name": null,
```

```
        "alert_matching_attribute": null,
        "components": [
            {
                "shingle_size": -1,
                "name": "agent",
                "treat_as": null,
                "similarity": 1.0
            },
            {
                "shingle_size": -1,
                "name": "description",
                "treat_as": null,
                "similarity": 0.75
            }
        ],
        "hop_limit": null,
        "version": "V2"
    },
    {
        "cookbooks": [
            "Default Cookbook"
        ],
        "name": "Source",
        "id": 2,
        "description": "Alerts from a similar source",
        "alert_threshold": 2,
        "trigger": "severity >= 3",
        "exclusion": "",
        "seed_alert": "",
        "rate": 0,
        "min_sample_size": 5,
        "max_sample_size": 10,
        "cook_for": null,
        "cook_for_extension": 0,
        "max_cook_for": null,
        "cluster_by": "first_match",
        "active": true,
        "chef": "CValueRecipeV2",
        "use_dynamic_topology": false,
        "topology_name": null,
        "alert_matching_attribute": null,
        "components": [
            {
                "shingle_size": 3,
                "name": "source",
                "treat_as": null,
                "similarity": 0.75
            },
            {
                "shingle_size": -1,
                "name": "agent",
                "treat_as": null,
                "similarity": 1.0
            }
        ],
        "hop_limit": null,
        "version": "V2"
    },
    {
        "cookbooks": [
            "Default Cookbook"
```

```
        ],
        "name": "Criticals",
        "id": 3,
        "description": "Remaining critical alerts",
        "alert_threshold": 1,
        "trigger": "severity = 5",
        "exclusion": "",
        "seed_alert": "",
        "rate": 0,
        "min_sample_size": 5,
        "max_sample_size": 10,
        "cook_for": null,
        "cook_for_extension": 0,
        "max_cook_for": null,
        "cluster_by": "first_match",
        "active": true,
        "chef": "CValueRecipeV2",
        "use_dynamic_topology": false,
        "topology_name": null,
        "alert_matching_attribute": null,
        "components": [
            {
                "shingle_size": 3,
                "name": "source",
                "treat_as": null,
                "similarity": 0.75
            },
            {
                "shingle_size": -1,
                "name": "agent",
                "treat_as": null,
                "similarity": 1.0
            }
        ],
        "hop_limit": null,
        "version": "V2"
    },
    {
        "cookbooks": [
            "BotCookBook1"
        ],
        "name": "BotRecipe2",
        "id": 4,
        "description": null,
        "alert_threshold": 2,
        "trigger": "",
        "exclusion": "",
        "seed_alert": "",
        "rate": 0,
        "min_sample_size": 5,
        "max_sample_size": 10,
        "cook_for": 3600,
        "cook_for_extension": 0,
        "max_cook_for": 0,
        "cluster_by": null,
        "active": true,
        "chef": "CBotRecipe",
```

```
        "use_dynamic_topology": false,
        "topology_name": null,
        "alert_matching_attribute": null,
        "initialize_function": "initBuckets",
        "member_function": "checkBucket",
        "use_in_recipe": null,
        "can_start_cluster": null,
        "similarity": 0.8
    },
    {
        "cookbooks": [
            "BotCookBook1"
        ],
        "name": "BotRecipe1",
        "id": 5,
        "description": null,
        "alert_threshold": 2,
        "trigger": "",
        "exclusion": "",
        "seed_alert": "",
        "rate": 0,
        "min_sample_size": 5,
        "max_sample_size": 10,
        "cook_for": 3600,
        "cook_for_extension": 0,
        "max_cook_for": 0,
        "cluster_by": null,
        "active": true,
        "chef": "CBotRecipe",
        "use_dynamic_topology": false,
        "topology_name": null,
        "alert_matching_attribute": null,
        "initialize_function": "initBuckets",
        "member_function": "checkBucket",
        "use_in_recipe": null,
        "can_start_cluster": null,
        "similarity": 0.8
    },
    {
        "cookbooks": null,
        "name": "GrazeRecipe1",
        "id": 6,
        "description": null,
        "alert_threshold": 2,
        "trigger": "",
        "exclusion": "",
        "seed_alert": "",
        "rate": 0,
        "min_sample_size": 5,
        "max_sample_size": 10,
        "cook_for": 3600,
        "cook_for_extension": 0,
        "max_cook_for": 0,
        "cluster_by": null,
        "active": true,
        "chef": "CValueRecipeV2",
        "use_dynamic_topology": false,
        "topology_name": null,
        "alert_matching_attribute": null,
        "components": [
            {
```

```
                "shingle_size": -1,
                "name": "custom_1",
                "treat_as": null,
                "similarity": 0.2
            }
        ],
        "hop_limit": null,
        "version": "V2"
    }
]
```

getResolvingThreadEntries

A Graze API GET request that returns thread entries for a specified Situation that have been marked as resolving steps. Threads are comments or 'story activity' on Situations. Operators can mark one or more thread entries as steps that were used to resolve a Situation.

You can select specific thread entries to return using **`start`** and **`limit`** values. If not, their default values return the first 100 thread entries. The thread entries returned are ordered by most recent first.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getResolvingThreadEntries`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.
`start`	Number	No	Number of the first resolving thread entry to return. Default is 0.
`limit`	Number	No	Maximum number of resolving thread entries to return. Default is 100.

Response

Endpoint **`getResolvingThreadEntries`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`sitn_id`	Number	Situation ID.
`resolving_entries`	Array	A list of resolving thread entries in the Situation. See below.

The **`resolving_entries`** list contains the following:

Name	Type	Description
`entry_text`	String	Text of the resolving entry. Reserved characters are converted to HTML entities, for example, **`&`** is converted to **`&`**.

user_id	Number	ID of the user that created the resolving entry.
thread_name	String	Name of the thread that the resolving entry is in.
time	Number	Timestamp (in Unix epoch time) of when the resolving entry was created.
entry_id	Number	ID of the resolving thread entry.

Examples

The following examples demonstrate typical use of endpoint **getResolvingThreadEntries**:

Request example

Example cURL request to return the first 100 resolving thread entries in Situation 358:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getResolvingThreadEntries" \
--data-urlencode "sitn_id=358"
```

Example cURL request to return the first 10 resolving thread entries in Situation 358:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getResolvingThreadEntries" \
--data-urlencode "sitn_id=358" \
--data-urlencode "start=0" \
--data-urlencode "limit=10"
```

Response example

Example response showing the three resolving thread entries in Situation 358:

```
{
   "sitn_id":358,
   "resolving_entries":
   [
      {
"entry_text":"hah","user_id":3,"thread_name":"Support","time":1549387456,"entry_
id":1722 },
      {
"entry_text":"asdfdsf","user_id":3,"thread_name":"Support","time":1549385762,"en
try_id":1721 },
      {
"entry_text":"sdfsdf\n","user_id":3,"thread_name":"Support","time":1549385747,"e
ntry_id":1720 }
   ]
}
```

getSecurityRealm

A Graze API GET request that returns a JSON object containing the names and configuration details of active SAML security realms.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSecurityRealm** takes the following request argument:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the

			authenticate endpoint for more information.

Endpoint **`getSecurityRealm`** takes no other arguments because this endpoint returns data on all active security realms.

Response

Endpoint **`getSecurityRealm`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
JSON Object	JSON	The name of the created Security Realms followed by its configured variables.

Examples

The following examples demonstrate typical use of endpoint **`getSecurityRealm`**:

Request example

Example cURL request to return the configuration of any active security realm in Cisco Crosswork Situation Manager:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSecurityRealms"
```

Response example

Successful requests return a JSON object representing the active realms. The following example shows a test SAML realm:

```
{
    "Test Saml Realm":
    {
        "configuration":
        {
            "defaultGroup":"EndUser",
            "realmType":"SAML2",
            "existingUserMappingField":"username",

"spMetadataFile":"/usr/share/moogsoft/config/keycloak.my_sp_metadata.xml",
            "defaultRoles":["Operator"],
            "defaultTeams":["Cloud DevOps"],
            "fullname":"$FirstName $LastName",
            "email":"$Email",

"idpMetadataFile":"/usr/share/moogsoft/config/keycloak.my_idp_metadata.xml",
            "username":"$Email",
            "maximumAuthenticationLifetime":60
        },
        "name":"Test Saml Realm",
        "active":true,
        "type":"SAML2"
```

```
    }
}
```

getServices

A Graze API GET request that returns a list of the services in the database.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getServices`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`limit`	Integer	No	Maximum number of services to return. Defaults to 1,000.
`start`	Integer	No	Number of the first service to return. Defaults to 0.
`query`	String	Yes	A substring match where the service name contains the query string.
`exact_match`	Boolean	No	If **`true`**, the query performs an exact match on the service name. If **`false`**, the query checks for **`contains`** only on the service name. Defaults to **`false`**.

Response

Endpoint **`getServices`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`service_id`	Number	ID of the service.
`name`	String	Service name.
`description`	String	Description of the service.

Examples

The following examples demonstrate typical use of endpoint **`getServices`**:

Example using exact matching

Example cURL request using exact matching of the query "Network LON":

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getServices" \
--data-urlencode 'query=Network LON' \
--data-urlencode 'exact_match=true'
```

Example response returning details of the service name "Network LON":

```
[{
   "service_id":3,
   "name":"Network LON",
```

```
   "description":"Network description"
}]
```

Example using approximate matching

Example cURL request using approximate matching of the query "Network":

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getServices" --data-
urlencode 'query=Network'
```

Example response returning details of all service names containing "Network":

```
[{
   "service_id":1,
   "name":"Network LON",
   "description":"Network London"
},{
   "service_id":2,
   "name":"NY Network A",
   "description":"Network New York A"
},{
   "service_id":3,
   "name":"NY Network B",
   "description":"Network New York B"
}]
```

getSeverities

A Graze API GET request that returns a list of possible severities and their severity IDs.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSeverities** takes the following request argument:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

Endpoint **getSeverities** takes no other arguments because this endpoint returns data on all severities.

Response

Endpoint **getSeverities** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	Severity name.
severity_id	Number	ID of the severity.

Examples

The following examples demonstrate typical use of endpoint **`getSeverities`**:

Request example

Example cURL request to return the list of all severities:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSeverities"
```

Response example

Example response returning a list of all severities:

```
[
   {
        "name": "Clear",
        "severity_id": 0
   },
   {
        "name": "Indeterminate",
        "severity_id": 1
   },
   {
        "name": "Warning",
        "severity_id": 2
   },
   {
        "name": "Minor",
        "severity_id": 3
   },
   {
        "name": "Major",
        "severity_id": 4
   },
   {
        "name": "Critical",
        "severity_id": 5
   }
]
```

getSigCorrelationInfo

A Graze API GET request that returns all correlation information related to a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSigCorrelationInfo`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.

Response

Endpoint **`getSigCorrelationInfo`** returns the following response:

Type	Description

HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
sig_id	Number	Situation ID.
service_name	String	Service name.
external_id	String	External ID.
properties	List of Strings	Properties of the Situation.

Examples

The following examples demonstrate typical use of endpoint **getSigCorrelationInfo**:

Request example

Example cURL request to return the correlation information for Situation ID 5:

```
curl -X GET -u graze:graze -k -v
"https://localhost/graze/v1/getSigCorrelationInfo?sitn_id=5" \
-H "Content-Type: application/json; charset=UTF-8"
```

Response example

Example response returning :

```
[
    {
        "sig_id": 1,
        "service_name": "Example1",
        "external_id": "Example1",
        "properties": "{"Example1":"Example1"}
    },
    {
        "sig_id": 2,
        "service_name": "Example2",
        "external_id": "Example2",
        "properties": "{"Example2":"Example2"}
    }
]
```

getSimilarSituationIds

A Graze API GET request that returns a list of IDs of similar Situations, for a specified Situation ID and a limit.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSimilarSituationIds** takes the following request arguments:

Name	Type	Required	Description

auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	ID of the Situation that you want to retrieve similar Situations for.
limit	Number	No	Maximum number of Situation IDs to return. Defaults to 100.

Response

Endpoint **getSimilarSituationIds** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
similarity_ids	Array of Numbers	List of IDs of similar Situations.
sig_id	Number	ID of the Situation that you requested to retrieve similar Situations for.

Examples

The following examples demonstrate typical use of endpoint **getSimilarSituationIds**:

Request example

Example cURL request to return the first 10 Situation IDs that are similar to Situation ID 1043:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSimilarSituationIds"
\
--data-urlencode "sitn_id=1043" \
--data-urlencode "limit=10"
```

Response example

Example response returning the Situation IDs that are similar to Situation ID 1043:

```
{
    "similarity_ids": [ 43,156,177,221,576,1026,1327 ],
    "sig_id": 1043
}
```

getSimilarSituations

A Graze API GET request that returns the details of similar Situations for a specified Situation and a limit.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSimilarSituations** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

sitn_id	Number	Yes	ID of the Situation that you want to retrieve similar Situations for.
limit	Number	No	Maximum number of Situations to return. Default is 100.

Response

Endpoint **getSimilarSituations** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
similarities	Array	A list of Situations with similarity details. For each similar Situation: **sim_sig_id**: ID of the similar situation. **similarity**: Similarity value. **resolving_steps_count**: Number of resolving steps that the similar Situation has.
similar_count	Number	Number of similar Situations.
sig_id	Number	ID of the Situation that you requested to retrieve similar Situations for.

Examples

The following examples demonstrate typical use of endpoint **getSimilarSituations**:

Request example

Example cURL request to return Situations that are similar to Situation ID 38:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSimilarSituations" \
--data-urlencode "sitn_id=38" \
--data-urlencode "limit=20"
```

Response example

Example response returning details of two Situations, IDs 39 and 40, that are similar to Situation ID 38:

```
{
    "similarities":[
    {
        "sim_sig_id":39,
        "similarity":1.0,
        "resolving_steps_count":0
    },
    {
        "sim_sig_id":40,
        "similarity":1.0,
        "resolving_steps_count":0
    }],
    "similar_count":2,
```

```
    "sig_id":38
}
```

getSituationActions

A Graze API GET request that returns the actions for Situations, ordered most recent last. You can use the **`from`** and **`to`** arguments to specify a period that you want to retrieve Situation actions for. If you do not specify these, actions for all dates and times are returned.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSituationActions`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_ids`	Array of Numbers	No, if you specify **`from`** and **`to`**.	Array of Situation IDs that the actions are requested for.
`start`	Integer	Yes	Number of the first action to return.
`limit`	Integer	Yes	Maximum number of actions that you want to return.
`actions`	Array of Numbers	No	List of action codes. Returns all action codes if no action codes are specified. See Situation Action Codes for a list of action codes and their descriptions.
`from`	Number	No	Start time of the period you want to retrieve Situation actions for. This is a Unix epoch timestamp in seconds.
`to`	Number	No	End time of the period you want to retrieve Situation actions for. This is a Unix epoch timestamp in seconds.

Response

Endpoint **`getSituationActions`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`activities`	Array	An array of objects containing the Situation action details.

Examples

The following examples demonstrate typical use of endpoint **`getSituationActions`**:

Request examples

Example cURL request to retrieve the first three actions for Situations 1, 2, 3 and 6:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationActions" \
--data-urlencode "sitn_ids=[1, 2, 3, 6]" \
```

```
--data-urlencode "actions=[1]" \
--data-urlencode "limit=3" \
--data-urlencode "start=0"
```

Example cURL command to retrieve the first 50 actions for Situations 1, 2, 3 and 6 between Unix epoch times 1553861746 and 1553872546:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationActions" \
--data-urlencode "sitn_ids=[1, 2, 3, 6]" \
--data-urlencode "actions=[1]" \
--data-urlencode "limit=50" \
--data-urlencode "start=0" \
--data-urlencode "from=1553861746"
--data-urlencode "to=1553872546"
```

Response example

Example response returning the actions for Situation 451:

```
{
    "activities": [
    {
        "uid": 2,
        "action_code": 1,
        "description": "Situation Created",
        "details": {},
        "type": "event",
        "sig_id": 451,
        "timed_at": 1507039842
    },
    {
        "uid": 2,
        "action_code": 14,
        "description": "Added Alerts To Situation",
        "details": {}
        "alerts": [1, 2]
    },
    {
        "uid": 3,
        "action_code": 11,
        "description": "Ran Tool",
        "details":
        {
        "tool_id": 271,
        "tool": "get data"
        },
        "sig_id": 451,
        "type": "event",
        "timed_at": 1557321088,
        "username": "admin"
    }]
}
```

getSituationAlertIds

A Graze API GET request that returns the total number of alerts, and a list of the alert IDs, for a specified Situation. This can be either all alerts or just those alerts unique to the Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSituationAlertIds** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.
for_unique_alerts	Boolean	No	Indicates the alerts to return from the Situation:**true**: Return only alerts unique to the Situation.**false**: Return all alerts in the Situation. Default.

Response

Endpoint **getSituationAlertIds** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
total_alerts	Number	Total number of alerts, or unique alerts.
alert_ids	Number list	A list of the alert IDs.

Examples

The following examples demonstrate typical use of endpoint **getSituationAlertIds**:

Request example

Example cURL request to return all the alert IDs for Situation ID 362:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationAlertIds" /
--data-urlencode "sitn_id=362" \
--data-urlencode "for_unique_alerts=false"
```

Response example

Example response returning all the alerts for Situation ID 362:


```
{"total_alerts":232,"alert_ids":[6,10,17,19,22,26,27,29,32,43,44,45,47,52,67,68,
79,81,83,84,96,102,105,108,109,111,113,115,116,125,135,136,138,140,142,143,147,1
51,152,153,165,175,177,178,180,181,188,192,193,207,211,213,217,223,225,232,238,2
39,240,244,255,258,259,269,270,272,274,284,293,303,314,318,335,357,363,369,374,3
75,388,398,414,428,430,434,442,443,448,449,450,479,480,485,486,492,494,504,505,5
10,511,518,521,529,556,558,563,570,580,594,596,599,601,603,628,655,656,661,664,6
74,684,691,705,714,715,719,720,728,732,734,750,776,777,781,788,794,808,819,830,8
35,838,844,857,858,860,861,877,882,885,887,890,892,893,900,901,906,912,914,918,9
26,936,937,959,971,972,984,994,1004,1013,1016,1019,1020,1023,1033,1043,1045,1068
,1076,1082,1083,1085,1099,1119,1124,1135,1137,1143,1147,1171,1185,1201,1207,1217
,1225,1231,1238,1254,1271,1272,1274,1280,1282,1290,1292,1301,1320,1321,1322,1324
,1326,1327,1331,1332,1333,1362,1379,1402,1414,1423,1433,1443,1454,1468,1472,1473
,1481,1491,1510,1512,1517,1520,1522,1532,1534]}
```

getSituationDescription

A Graze API GET request that returns the description for a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSituationDescription`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.

Response

Endpoint **`getSituationDescription`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`sitn_id`	Number	Situation ID.
`description`	String	Text in the Situation' description field.

Examples

The following examples demonstrate typical use of endpoint **`getSituationDescription`**:

Request example

Example cURL request to return the description for Situation ID 231:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getSituationDescription" \
--data-urlencode "sitn_id=231"
```

Response example

Example response returning the description for Situation ID 231:

```
{
    "sitn_id": "231",
    "description": "SyslogLamCookbook source"
}
```

getSituationDetails

A Graze API GET request that returns the details of a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSituationDetails`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.

Response

Endpoint **`getSituationDetails`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`category`	String	Category of the alert.
`created_at`	Number	Time when Cisco Crosswork Situation Manager created the Situation. This is a Unix epoch timestamp in seconds.
`custom_info`	Object	Object containing the custom info for the Situation; **`null`** if there is no custom info for the Situation.
`description`	String	Description of the Situation.
`first_event_time`	Number	Time when Cisco Crosswork Situation Manager received the first event. This is a Unix epoch timestamp in seconds.
`internal_priority`	Number	Internal priority of the Situation.
`last_event_time`	Number	Time when Cisco Crosswork Situation Manager received the latest event. This is a Unix epoch timestamp in seconds.
`last_state_change`	Number	Time when the last state change occurred. This is a Unix epoch timestamp in seconds.
`moderator_id`	String	Owner of the Situation.
`sitn_id`	Number	Situation ID.
`status`	Number	Status of the Situation.
`story_id`	Number	
`superseded_by`	String	The ID of the Situation that supersedes this Situation, **`null`** if the Situation is not superseded.
`total_alerts`	Number	Total number of alerts in the Situation.
`total_unique_alerts`	Number	Total number of alerts that are unique to the Situation.
`primary_team_id`	Number	ID of the primary team assigned to the Situation. This is not returned if there is no primary team.

Examples

The following examples demonstrate typical use of endpoint **`getSituationDetails`**:

Request example

Example cURL request to the details of Situation ID 173:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationDetails" \
--data-urlencode "sitn_id=173"
```

Response example

Example response returning the details of Situation ID 173:


```
{
    "category":"Detected",
    "created_at":1415814620,
    "custom_info":null,
    "description":"Sigaliser situation",
    "first_event_time":1415814600,
    "internal_priority":0,
    "last_event_time":1415814619,
    "last_state_change":1415868947,
    "moderator_id":2,
    "sitn_id":173,
    "status":1,
    "story_id":3,
    "superseded_by":null,
    "total_alerts":1403,
    "total_unique_alerts":1403,
    "primary_team_id":2
}
```


getSituationFlags

A Graze API GET request that returns the flags for one or an array of Situations.

See Situation Flags for more information on Cisco Crosswork Situation Manager Situation flags.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSituationFlags`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_ids`	Array of Numbers	Yes	A list of Situation IDs.

Response

Endpoint **`getSituationFlags`** returns the following response:

Type	Description

HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Description
`JSON List`	List of the states those Situations

Examples

The following examples demonstrate typical use of endpoint **`getSituationFlags`**:

Request example

Example cURL request to list an array of all flags associated with Situation 1.

```
curl -X GET -u graze:graze -k -v
https://localhost/graze/v1/getSituationFlags?sitn_ids=%5B1%5D
```

Response example

Example response returning the flags associated with Situation 1:

```
{
    "1":
    [
        "NOTIFIED",
        "TICKETED"
    ]
}
```

getSituationHosts

A Graze API GET request that returns the hosts for a specified Situation, either for all the alerts in the Situation or just for the unique alerts.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSituationHosts`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.
`for_unique_alerts`	Boolean	No	Indicates the host names to return from the Situation:**`true`**: Return only host names unique to the Situation.**`false`**: Return all host names in the Situation. Default.

Response

Endpoint **`getSituationHosts`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
hosts	JSON object	An array of all hosts that sent alerts contained in the specified Situation.

Examples

The following examples demonstrate typical use of endpoint **getSituationHosts**:

Request example

Example cURL request to return all the hosts that sent alerts to Situation ID 447:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationHosts" \
--data-urlencode "sitn_id=447"
```

Response example

Example response returning all the hosts that sent alerts to Situation ID 447:

```
{
    hosts:
    [
        "xldn1204pap",
        "xldn1215pap",
        "xldn1220pap",
        "vxldn1230pap",
        "xldn1241pap",
        "xldn1252pap",
        "xldn1271pap",
        "xldn1278pap",
        "xldn1297pap",
        "xldn1299pap"
     ]
}
```

getSituationIds

A Graze API GET request that returns the total number of Situations, and a list of their Situation IDs, for a specified Situation filter and a limit.

Note

Take special care when using endpoint **getSituationIds**. Overuse of this endpoint can have a negative impact on the backend datastore.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSituationIds** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
query	String	Yes	An SQL-like filter that Situations must match to be returned.
limit	Number	No	Maximum number of Situation IDs to return.

Response

Endpoint **getSituationIds** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
total_situations	Number	Total number of Situations, or unique Situations.
sitn_ids	List	A list of Situation IDs.

Examples

The following examples demonstrate typical use of endpoint **getSituationIds**:

Request example

Example cURL request to get the first 20 Situation IDs that match the query where the description is "lon_storage_636728" or the queue is 5:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationIds" \
--data-urlencode 'query=description="lon_storage_636728" or queue = 5' \
--data-urlencode 'limit=20'
```

Response example

Example response returning seven Situation IDs that match the query:

```
{
    "total_situations":7,
    "sitn_ids":[87,121,128,278,523,1003,1519]
}
```

getSituationPrimaryTeam

A Graze API GET request that returns the primary team on the specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSituationPrimaryTeam** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	ID of the Situation you want to retrieve the primary team for.

Response

Endpoint **getSituationPrimaryTeam** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
sitn_id	Number	ID of the Situation you wanted to retrieve the primary team for.
primary_team_id	Number	ID of the primary team for the Situation.

Examples

The following examples demonstrate typical use of endpoint **getSituationPrimaryTeam**:

Request example

Example cURL request to return the primary team for Situation 1906:

```
curl -G -u graze:graze -k "https://localhost/graze/v1/getSituationPrimaryTeam" \
--data-urlencode 'sitn_id=1906'
```

Response examples

Example response returning that team 36 is the primary team for Situation 1906:

```
{
    "primary_team_name": "Cloud DevOps",
    "sitn_id": 1906,
    "primary_team_id": 1
}
```

Example response returning that Situation 1906 does not have a primary team assigned to it:

```
{
    "sitn_id":1906,
}
```

getSituationProcesses

A Graze API GET request that returns a list of process names for a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSituationProcesses** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	ID of the Situation you want to return the process names for.

Response

Endpoint **getSituationProcesses** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
processes	Array	A list of all the Situation's process names.

Examples

The following examples demonstrate typical use of endpoint **getSituationProcesses**:

Request example

Example cURL request to return all the process names for Situation 473:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationProcesses"
\
--data-urlencode "sitn_id=473"
```

Response example

Example response returning a list of all the Situation's process names:

```
{
    "processes":
    [
        "Knowledge Management",
        "Online Transaction Processing",
        "Web Content Management",
        "40GbE",
        "8-bit Unicode Transcoding Platform"
    ]
}
```

getSituationServices

A Graze API GET request that returns a list of external service names for a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSituationServices** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.

Response

Endpoint **getSituationServices** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
services	Array	A list of the Situation's services.

Examples

The following examples demonstrate typical use of endpoint **getSituationServices**:

Request example

Example cURL request to return the services for Situation ID 345:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationServices" \
--data-urlencode "sitn_id=345"
```

Response example

Example response returning the services for the specified Situation:

```
{
   "services":
    [
       "Cloud Management Platform",
       "Geographic Information Systems",
       "Knowledge Management",
       "Online Transaction Processing",
       "Storage Subsystem",
       "Web Content Management",
       "0-bit Emulation","40GbE",
       "8-bit Unicode Transcoding Platform"
   ]
}
```

getSituationSeverityChanges

A Graze API GET request that returns the changes in severity for a Situation. The highest severity of any of the alerts in a Situation determines the severity of the Situation. This endpoint returns increases in severity and a change to a severity of 0 (Clear). If a Situation has closed, this endpoint returns a severity of 0 (Clear) and the timestamp when the Situation was closed. The endpoint does not return any further changes in severity after it has returned to 0 (Clear).

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSituationSeverityChanges`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	ID of the Situation you want to return severity changes for.

Response

Endpoint **`getSituationSeverityChanges`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description

timestamp	Number	The time when the severity change occurred. This is a Unix epoch timestamp in seconds.
severity	Number	The new severity of the Situation: 0 = Clear1 = Indeterminate2 = Warning3 = Minor4 = Major5 = Critical

Examples

The following examples demonstrate typical use of endpoint **getSituationSeverityChanges**:

Request example

Example cURL request to return the severity changes for Situation ID 234:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getSituationSeverityChanges" \
--data-urlencode "sitn_id=234"
```

Response example

Example response returning the severity changes for the Situation. The response shows increases in severity and the change to a severity of 0 (Clear).

```
[
  {
    "timestamp": 1580193608,
    "severity": 4
  },
  {
    "timestamp": 1580193660,
    "severity": 5
  },
  {
    "timestamp": 1580193667,
    "severity": 0
  }
]
```

getSituationsWithFlag

A Graze API GET request that returns all the Situations which have the specified flag.

See Situation Flags for more information on Cisco Crosswork Situation Manager Situation flags.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getSituationsWithFlag** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
flag	String	Yes	Name of the flag to search for.
start	Number	No	Starting point of the result set to return. Default is 0.
limit	Number	No	Maximum number of results to return. Default is 1000.

Response

Endpoint **`getSituationsWithFlag`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Type	Description
JSON Array	An array of the Situation IDs that have the specified flag associated with them.

Examples

The following examples demonstrate typical use of endpoint **`getSituationsWithFlag`**:

Request example

Example cURL request to retrieve all Situations that have the specified flag associated with them.

```
curl -X GET -u graze:graze -k -v
https://localhost/graze/v1/getSituationsWithFlag?flag=NOTIFIED
```

Response example

Example response returning an array of all of the Situations that have the specified flag associated with them:

```
[ 1, 2, 5 ]
```

getSituationTopology

A Graze API endpoint that returns the topology details for a specified Situation and topology. The request returns a JSON object that lists the links and nodes affected by the Situation in a specified topology. It also returns the alert matching attributes for the nodes in the topology.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSituationTopology`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.
`topology_name`	String	Yes	Name of the topology for which to return the Situation's link, node and alert matching attribute details. A Situation can impact nodes in multiple topologies.
`context`	Number	Yes	Number, between 0 and 4, of contextual hops from the nodes directly affected within the Situation to other nodes to be included in the returned object. See Vertex Entropy for more information on contextual hops.Vertex Entropy**`0`**: Only nodes directly affected by the Situation. Default.**`4`**: Nodes that are up to

			four hops away from the nodes directly affected by the Situation.
properties	Array of Strings	Yes	List of the node properties to be returned. Valid properties are: **severity**: Severity of the node. **prc**: Whether this node is the probable root cause of the alert. **description**: Description of the node. **vertex_entropy**: Vertex Entropy of the node. See Vertex Entropy for more information.

Response

Endpoint **getSituationTopology** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
links	Array of strings	One or more links associated with the Situation, including the following properties: **source**: Source node of the link. **target**: Target node of the link. Note that links in Cisco Crosswork Situation Manager are bidirections.
nodes	Array of strings	One or more nodes associated with the Situation and their IDs and properties. The **context** request property determines which nodes are included.
alertMatchingAttributes	Array of strings	The alert fields that specify the topology nodes from which the alerts were generated.

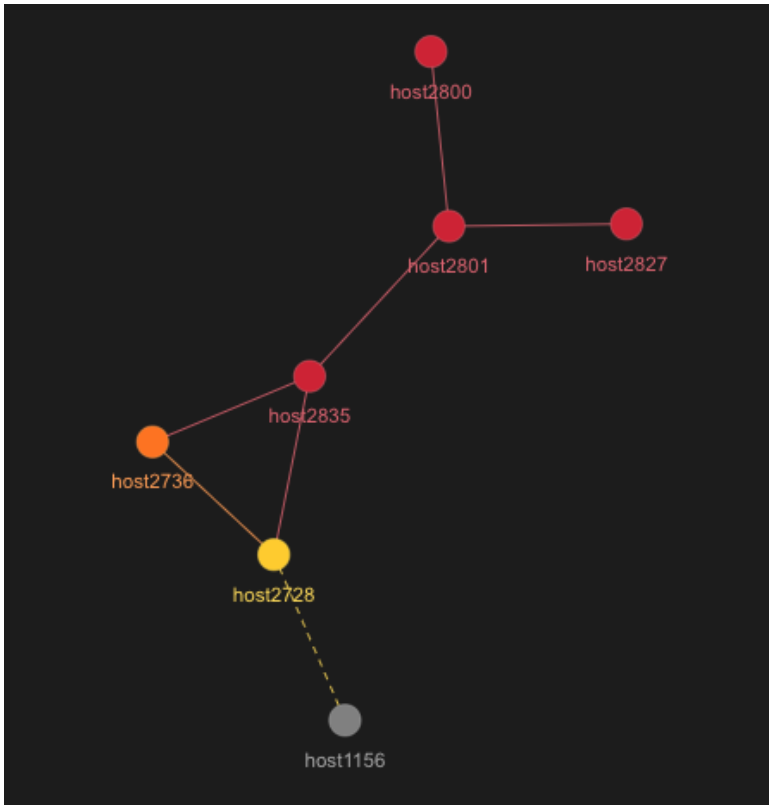
See http://netjson.org/ for more information on the topology data format.

Examples

The following examples demonstrate typical use of endpoint **getSituationTopology**:

Request example

The following topology diagram shows the nodes affected by Situation ID 14, with a context of **1**. In this example, each node represents a host in a network and the Situation represents a network outage. It shows six nodes directly affected by the Situation, their color depending on their severity, and one node which is one hop away, shown in gray.



The following cURL request demonstrates a request to return nodes affected by the Situation and nodes that are one hop away in the "network" topology. The returned object contain the properties of severity, Vertex Entropy, Probable Root Cause (PRC), and description.

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSituationTopology" \
--data-urlencode "sitn_id=14" \
"topology_name=network" \
"context=1" \
"properties"= [ "severity","vertex_entropy","prc","description" ]
```

Response example

The successful response returns the following topology information for this Situation. Note that there is no PRC value for the node that is not directly affected by the Situation. In this example, consider investigating node "host2835" as the cause of the Situation because it has a high severity and a high PRC.

```
{
    "links":
    [
        {
            "source": "host2728",
            "target": "host2736"
        },
        {
            "source": "host2728",
            "target": "host1156"
        },
        {
```

```
            "source": "host2835",
            "target": "host2728"
        },
        {
            "source": "host2801",
            "target": "host2827"
        },
        {
            "source": "host2800",
            "target": "host2801"
        },
        {
            "source": "host2801",
            "target": "host2835"
        },
        {
            "source": "host2835",
            "target": "host2736"
        }
    ],
    "nodes":
    [
        {
            "id": "host2835",
            "properties": {
                "severity": 5,
                "prc": 0.9862626716344282,
                "context": 0,
                "description": "node1",
                "vertex_entropy": 0.1794592472207979
                }
            },
        {
            "id": "host2736",
            "properties": {
                "severity": 4,
                "prc": 0.42722191049803876,
                "context": 0,
                "description": "node2",
                "vertex_entropy": 0.08976540495989357
                }
            },
        {
            "id": "host2728",
            "properties": {
                "severity": 3,
                "prc": 0.007672752075071621,
                "context": 0,
                "description": "node3",
                "vertex_entropy": 0.1794592472207979
            }
        },
        {
            "id": "host2827",
            "properties": {
                "severity": 5,
                "prc": 0.4262762946261391,
                "context": 0,
                "description": "node4",
                "vertex_entropy": 0.05343516483103129
            }
```

```
            },
            {
                "id": "host2801",
                "properties": {
                    "severity": 5,
                    "prc": 0.42722511225514104,
                    "context": 0,
                    "description": "node5",
                    "vertex_entropy": 0.23927899629439717
                }
            },
            {
                "id": "host2800",
                "properties": {
                    "severity": 5,
                    "prc": 0.4269879766269776,
                    "context": 0,
                    "description": "node6",
                    "vertex_entropy": 0.05343516483103129
                }
            },
            {
                "id": "host1156",
                "properties": {
                    "severity": null,
                    "prc": null,
                    "context": 1,
                    "description": "node7",
                    "vertex_entropy": 0.05343516483103129
                }
            }
        ],
        "alertMatchingAttributes": ["source"]
    }
```


getSituationVisualization

A Graze API GET request that returns the Visualize information for a Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSituationVisualization`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.

Response

Endpoint **`getSituationVisualization`** returns the following response:

Type	Description
HTTP	HTTP status or error code indicating request success or failure. See HTTP status code

Code	definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
sig_id	Integer	Situation ID.
origin	String	Process that caused the Situation to be created, for example, **cookbook** or **manual_merge**.
cause	Object	Details of the origin of the Situation, for example the **cookbook_name**, **recipe_id**, **cookbook_id**, **recipe_name**, **reference_alert_id** and **reference_event_id**.
thresholds	Object	The **saved** and **original** threshold values for the Situation, if the Recipe has been updated. If a threshold was removed from the Recipe after Situation creation, it is not returned in the response. If a threshold was added to the Recipe after Situation creation, the **saved** and **original** values are returned.

Examples

The following examples demonstrate typical use of the **getSituationVisualization** endpoint:

Request example

Example cURL request to return information on the origin and cause of Situation ID 358:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getSituationVisualization" --data-urlencode
"sitn_id=358"
```

Response examples

Example response for a Situation created by a Cookbook Recipe:

```
{
    "thresholds":
    {
        "agent":
        {
            saved: 1.0,
            original: 0.7
        },
        "source":
        {
            saved: 0.97,
            original: 0.6
        }
    },
    "origin": "Cookbook",
    "cause":
    {
        "cookbook_name": "Default Cookbook",
        "recipe_id": 5,
        "cookbook_id": 7,
        "recipe_name": "Source",
        "reference_alert_id": 198,
        "reference_event_id": 210
    },
```

```
    "sig_id":5
}
```

Example response for a manually created Situation:

```
{
    "origin": "Manual Creation",
    "cause": {"uid": 3},
    "sig_id": 6
}
```

Example response when two Situations have been merged:

```
{
    "origin": "Manual Merge",
    "cause":
    {
        "uid": 3,
        "merged_sigs": [8,7]
    },
    "sig_id": 9
}
```

If there is no Situation visualization data, the response returns the following information:

```
{
    "additional":
    {
        "debugMessage": "com.moogsoft.servletutils.CGeneralServerException:
com.moogsoft.services.CGeneralServiceException: No visualize data found for
Situation ID [2323]"
    },
    "message": "Internal server error",
    "statusCode": 1000
}
```

getStatuses

A Graze API GET request that returns a list of statuses that can apply to Situations and their IDs.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getStatuses`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

There are no other arguments because this endpoint returns data on all statuses.

Response

Endpoint **`getStatuses`** returns the following response:

Type	Description
HTTP	HTTP status or error code indicating request success or failure. See HTTP status code

Code	definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
status_id	Number	ID of the status.
name	String	Status name.

Examples

The following examples demonstrate typical use of endpoint **getStatuses**:

Request example

Example cURL request to return a list of statuses:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getStatuses"
```

Response example

Example response returning a list of statuses:

```
[
   {
        "status_id": 1,
        "name": "Opened"
   },
   {
        "status_id": 2,
        "name": "Unassigned"
   },
   {
        "status_id": 3,
        "name": "Assigned"
   },
   {
        "status_id": 4,
        "name": "Acknowledged"
   },
   {
        "status_id": 5,
        "name": "Unacknowledged"
   },
   {
        "status_id": 6,
        "name": "Active"
   },
   {
        "status_id": 7,
        "name": "Dormant"
   },
   {
        "status_id": 8,
        "name": "Resolved"
   },
   {
        "status_id": 9,
        "name": "Closed"
   },
   {
        "status_id": 10,
```

```
            "name": "SLA Exceeded"
        }
    ]
```

getSystemStatus

A Graze API GET request that returns current system status information for all processes.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSystemStatus`** takes the following request argument:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

Endpoint **`getSystemStatus`** takes no other arguments because this endpoint returns data on all processes.

Response

Endpoint **`getSystemStatus`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`component`	String	Represents the name of a component within the process. May not be present, depending on the process.
`instance`	String	Instance name.
`last_heartbeat`	Number	Timestamp, in milliseconds, of the last process heartbeat. **`0`** is a special value indicating that a heartbeat has never been received.
`missed_heartbeats`	Number	Number of missed process heartbeats. **`-1`** is a special value indicating that a heartbeat has never been received.
`process_name`	String	Process name.
`processes`	Array	A list of the processes, with status information.
`reserved`	Boolean	Indicates whether the process is reserved: **`true`** = a reserved process **`false`** = process that is not reserved
`running`	Boolean	Indicates whether the process is running:

		true = running **false** = not running
service_name	String	Service name.
display_name	String	Name of the service in the configuration.
type	String	Type of service, for example, lam, servlet, Moogfarmd.
passive	Boolean	Indicates whether the service is passive in a HA environment: **true** = passive **false** = active
stoppable	Boolean	Indicates whether the service is passive can be stopped: **true** = stoppable **false** = not stoppable
ha_conf	JSON Object	A JSON blob containing the HA configuration.
additional_health_info	JSON Object	Additional health information. The **pools** section includes health information for processes with an internal pool.

Examples

The following examples demonstrate typical use of endpoint **getSystemStatus**:

Request example

Example cURL request to return the system status:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSystemStatus"
```

Response example

Example response returning the system status:

```
{
    "processes": [{
        "running": true,
        "sub_components": {
        "moogpoller": {
            "run_on_startup": true,
            "instance": "",
            "service_name": "apache-tomcat",
            "display_name": "servlets",
            "type": "servlets",
            "last_heartbeat": 1491385834300,
            "passive": false,
            "running": true,
            "component": "moogpoller",
            "reserved": true,
            "stoppable": true,
            "missed_heartbeats": 0,
            "ha_conf": {
                "cluster": "MOO",
                "instance": "",
```

```
                "default_leader": true,
                "start_as_passive": false,
                "only_leader_active": false,
                "group": "servlets"
            }
        },
    "moogsvr": {
        "run_on_startup": true,
        "instance": "",
        "service_name": "apache-tomcat",
        "display_name": "servlets",
        "type": "servlets",
        "last_heartbeat": 1491385825246,
        "passive": false,
        "running": true,
        "component": "moogsvr",
        "reserved": true,
        "stoppable": true,
        "missed_heartbeats": 0,
        "ha_conf": {
            "cluster": "MOO",
            "instance": "",
            "default_leader": true,
            "start_as_passive": false,
            "only_leader_active": false,
            "group": "servlets"
        }
      }
  },
"instance": "",
    "reserved": true,
    "service_name": "apache-tomcat",
    "stoppable": true,
    "missed_heartbeats": 0,
    "display_name": "servlets",
    "type": "servlets",
    "last_heartbeat": 1491385834300,
    "ha_conf": {
        "cluster": "MOO",
        "instance": "",
        "default_leader": true,
        "start_as_passive": false,
        "only_leader_active": false,
        "group": "servlets"
        },
    "passive": false
        }, {
            "running": false,
            "instance": "",
            "last_missed_heartbeat": 1491385820601,
            "reserved": false,
            "stoppable": false,
            "missed_heartbeats": 10,
            "display_name": "test_lam",
            "type": "lam",
            "last_heartbeat": 1491382820601,
            "additional_health_info": {
```

```
                        "thread_pool_queue_size": 0,
                        "published_events": {
                                "last_5_minutes": 130,
                                "last_10_minutes": 130,
                                "last_minute": 130
                        }
                },
                "ha_conf": {
                        "cluster": "MOO",
                        "instance": "",
                        "default_leader": true,
                        "start_as_passive": false,
                        "only_leader_active": true,
                        "group": "test_lam"
                },
                "passive": false
        "sub_components": {
            "SituationMgr": {
            "run_on_startup": true,
            "instance": "",
            "last_missed_heartbeat": 1491385821669,
            "service_name": "moogfarmd",
            "display_name": "moog_farmd",
            "type": "moog_farmd",
            "last_heartbeat": 1491382821669,
            "passive": false,
            "running": false,
            "component": "SituationMgr",
            "reserved": true,
            "stoppable": true,
            "missed_heartbeats": 10,
            "ha_conf": {
                "cluster": "MOO",
                "instance": "",
                "default_leader": true,
                "start_as_passive": false,
                "only_leader_active": true,
                "group": "moog_farmd"
             }
          },
    "AlertBuilder": {
        "run_on_startup": true,
        "instance": "",
        "last_missed_heartbeat": 1491385821669,
        "service_name": "moogfarmd",
        "display_name": "moog_farmd",
        "type": "moog_farmd",
        "last_heartbeat": 1491382821669,
        "passive": false,
        "running": false,
        "component": "AlertBuilder",
        "reserved": true,
        "stoppable": true,
        "missed_heartbeats": 10,
        "ha_conf": {
            "cluster": "MOO",
            "instance": "",
            "default_leader": true,
            "start_as_passive": false,
            "only_leader_active": true,
            "group": "moog_farmd"
```

```
            }
          },
        "TeamsMgr": {
            "run_on_startup": true,
            "instance": "",
            "last_missed_heartbeat": 1491385821669,
            "service_name": "moogfarmd",
            "display_name": "moog_farmd",
            "type": "moog_farmd",
            "last_heartbeat": 1491382821669,
            "passive": false,
            "running": false,
            "component": "TeamsMgr",
            "reserved": true,
            "stoppable": true,
            "missed_heartbeats": 10,
            "ha_conf": {
                "cluster": "MOO",
                "instance": "",
                "default_leader": true,
                "start_as_passive": false,
                "only_leader_active": true,
                "group": "moog_farmd"
            }
        }
    },
    "instance": "",
    "last_missed_heartbeat": 1491385821669,
    "service_name": "moogfarmd",
    "display_name": "moog_farmd",
    "type": "moog_farmd",
    "last_heartbeat": 1491382821669,
    "additional_health_info": {
        "event_processing_metric": 0.65
    },
"passive": false,
"running": false,
"reserved": true,
"stoppable": true,
"missed_heartbeats": 10,
"ha_conf": {
    "cluster": "MOO",
    "instance": "",
    "default_leader": true,
    "start_as_passive": false,
    "only_leader_active": true,
    "group": "moog_farmd"
    }
 },
{
    "running": false,
    "instance": "",
    "reserved": false,
    "service_name": "restclientlamd",
    "stoppable": true,
    "display_name": "rest_client_lam",
    "type": "lam",
```

```
        "ha_conf": {
            "cluster": "MOO",
            "instance": "",
            "group": "rest_client_lam"
         }
         "additional_health_info": {
             "pools": {
                 "MoogPoller": [{
                 "removed": 0,
                 "ration": 0.0,
                 "busy": 0,
                 "resource_type": "com.mysql.jdbc.JDBC4Connection",
                 "checkout_per_second": 0.0,
                 "free": 10,
                 "avg_checkedout_seconds": 0.0,
                 "capacity": 10
              }],
              "Message sender pool": [{
                  "removed": 0,
                  "ration": 0.0,
                  "busy": 0,
                  "resource_type": "com.moogsoft.mooms.CMoomsMessageSender",
                  "checkout_per_second": 0.09997000899730081,
                  "free": 10,
                  "avg_checkedout_seconds": 0.002,
                  "capacity": 10
                }]
            }
        }]
    }
```


getSystemSummary

A Graze API GET request that returns a summary of current alerts and Situations in Cisco Crosswork Situation Manager.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getSystemSummary`** takes the following request argument:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

Endpoint **`getSystemSummary`** takes no other arguments because this endpoint returns data on all alerts and Situations.

Response

Endpoint **`getSystemSummary`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object **`system_summary`**, containing the following statistics:

Name	Type	Description

total_events	Number	Total number of events in Cisco Crosswork Situation Manager.
open_sitns	Number	Number of open Situations in Cisco Crosswork Situation Manager.
open_sitns_up	Number	Number of open Situations that are trending up.
open_sitns_down	Number	Number of open Situations that are trending down.
avg_events_per_sitn	Number	Average number of events per Situation.
avg_alerts_per_sitn	Number	Average number of events per Situation.
service_count	Number	Number of services in Cisco Crosswork Situation Manager.
open_sigs_unassigned	Number	Number of unassigned Situations.

Examples

The following examples demonstrate typical use of endpoint **getSystemSummary**:

Request example

Example cURL request to return a summary of alerts and Situations in Cisco Crosswork Situation Manager:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getSystemSummary"
```

Response example

Example response returning a summary of alerts and Situations in Cisco Crosswork Situation Manager:

```
{
    "system_summary":
    {
        "total_events":61676,
        "open_sitns":571,
        "avg_events_per_sitn":305,
        "open_sitns_up":565,
        "open_sitns_down":2,
        "avg_alerts_per_sitn":16,
        "open_sigs_unassigned":310,
        "timestamp":1499425056
    }
}
```

getTeam

A Graze API GET request that returns a team's details by team ID or name.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getTeam** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See

			the authenticate endpoint for more information.
team_id	Integer	Yes	Unique ID of the team to retrieve information about.
name	String	Yes	Name of a valid team to retrieve information about.

Response

Endpoint **getTeam** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Type	Description
JSON Object	A JSON object containing details of the specified team.

Examples

The following examples demonstrate typical use of endpoint **getTeam**:

Request examples

Example cURL request to return details of the team ID 1:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getTeam?team_id=1"
```

Example cURL request to return details of the team "Cloud DevOps:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getTeam?name=Cloud
DevOps"
```

Response example

Example response returning details of the team:

```
{
  "room_id": 1,
  "alert_filter": "((severity = 0) OR (severity = 1)) AND (agent_location =
\"Test\")",
  "user_ids": [
    3
  ],
  "sig_filter": "((internal_priority = 0) AND (internal_priority = 1)) OR
(description = \"Test\")",
  "landing_page": "",
  "description": "",
  "active": true,
  "team_id": 1,
  "services": [],
  "users": [
    "admin"
  ],
  "deleted": false,
  "name": "Cloud DevOps",
  "service_ids": []
}
```

getTeams

A Graze API GET request that returns information on all the teams in Cisco Crosswork Situation Manager.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getTeams`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

Endpoint **`getTeams`** takes no other arguments because this endpoint returns data on all the teams in Cisco Crosswork Situation Manager.

Response

Endpoint **`getTeams`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Type	Description
JSON Object	Details of each team in Cisco Crosswork Situation Manager.

Examples

The following examples demonstrate typical use of endpoint **`getTeams`**:

Request example

Example cURL request to return all the teams in Cisco Crosswork Situation Manager:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getTeams"
```

Response example

Example response returning details of all the teams in Cisco Crosswork Situation Manager:

```
[
    {
        "room_id": 1,
        "alert_filter": "",
        "user_ids": [3],
        "sig_filter": "",
        "landing_page": "",
        "description": "Example Team",
        "active": true,
        "team_id": 1,
        "services": ["Commerce","Compute","CRM","Database"],
        "users": ["admin"],
        "name": "Cloud DevOps",
```

```
            "service_ids": [1,2,3,4]
        },
        {
            "room_id": 2,
            "alert_filter": "(description = \"Test\") AND ((severity = 0) OR
    (severity = 2))",
            "user_ids": [5,6],
            "sig_filter": "((internal_priority = 0) OR (internal_priority = 1)) AND
    (description = \"Test\")",
            "landing_page": "",
            "description": "Team based in Kingston",
            "active": true,
            "team_id": 2,
            "services": ["Kingston::AD::Server","Kingston::Application::Server"],
            "users": ["AnnaMatthews1","JorgeHowell2"],
            "deleted": false,
            "name": "Team Kingston",
            "service_ids": [1,2]
        }
    ]
```


getTeamsForService

A Graze API GET request to return all teams related to a service with the specified ID or name.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getTeamsForService`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`service_id`	String	No, if you specify **`name`**.	ID of the service.
`name`	String	No, if you specify **`service_id`**.	Name of the service.

Response

Endpoint **`getTeamsForService`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Type	Description
JSON Object	A JSON object containing details of each team related to the specified service.

Examples

The following examples demonstrate typical use of endpoint **`getTeamsForService`**:

Request examples

Example cURL requests to return the teams related to service ID 1:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getTeamsForService?service_id=1"
```

Example cURL requests to return the teams related to service `"web"`:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getTeamsForService?service_name=web"
```

Response example

Example response returning details of a team related to service `"web"`:

```
[
    {
        "room_id": 1,
        "alert_filter": "((severity = 0) OR (severity = 1)) AND (agent_location
= \"Test\")",
        "user_ids": [3],
        "sig_filter": "((internal_priority = 0) AND (internal_priority = 1)) OR
(description = \"Test\")",
        "name": "Cloud DevOps",
        "landing_page": "",
        "description": "Example Team",
        "active": true,
        "service_ids": [1,2,3,4],
        "team_id": 1,
        "services": ["Commerce","Compute","CRM","Database"],
        "users": ["admin"]
    }
]
```

getTeamSituationIds

Request that returns the total number of Situations that are assigned to a team, and a list of their Situation IDs.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getTeamSituationIds`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`team_name`	String	Yes	Name of an existing team.

Response

Endpoint **`getTeamSituationIds`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
total_situation	Number	The total number of Situations assigned to a team.
sitn_ids	Number list	A list of Situation IDs of the Situations assigned to a team.

Examples

The following examples demonstrate typical use of endpoint **getTeamSituationIds**:

Request example

Example cURL request to return the Situations assigned to team "Cloud Devops":

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getTeamSituationIds" \
--data-urlencode "team_name=Cloud Devops"
```

Response example

Example response returning the total number of Situations followed by the ID of each situation.

```
{
   "total_situations": 7,"sitn_ids": [20,21,39,55,85,119,145]
}
```

getTempus

A Graze API GET request that returns the details of all Tempus Moolets in Cisco Crosswork Situation Manager.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getTempus** takes the following request argument:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

Endpoint **getTempus** takes no other arguments because this endpoint returns data on all Tempus Moolets in Cisco Crosswork Situation Manager.

Response

Endpoint **getTempus** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Type	Description
JSON Object	Names and configurations of all Tempus Moolets in Cisco Crosswork Situation Manager.

Examples

The following examples demonstrate typical use of endpoint **getTempus**:

Request example

Example cURL request to return the details of all Tempus Moolets:

```
curl -G -u graze:graze -k "https://localhost/graze/v1/getTempus"
```

Response example

Example response returning the details of Tempus algorithm "Time Based (Tempus)":


```
[
    {
        "detection_algorithm": "Louvain",
        "minimum_arrival_similarity": 0.6667,
        "run_on_startup": true,
        "arrival_spread": 15,
        "execution_interval": 120,
        "description": "A Tempus Situation",
        "alert_threshold": 4,
        "pre_partition": null,
        "partition_by": null,
        "window_size": 1200,
        "edge_weight": false,
        "significance_threshold": 1,
        "name": "Time Based (Tempus)",
        "entropy_threshold": 0.0,
        "threshold_type": "global",
        "process_output_of": "Alert Workflows",
        "significance_test": "Poisson1",
        "bucket_size": 5
    }
]
```


getThreadEntries

A Graze API GET request that returns thread entries for a specified thread and Situation. Threads are comments or 'story activity' on Situations.

You can request to return specific thread entries using **`start`** and **`limit`** values. If not, their default values return the first 100 entries. The entries returned are ordered by most recent first.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getThreadEntries`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.
`thread_name`	String	Yes	Name of the thread to get entries from.
`start`	Number	No	Number of the first thread entry to return. Default is 0.
`limit`	Number	No	Maximum number of thread entries to return. Default is 100.

Response

Endpoint **`getThreadEntries`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`entries`	List	A list of thread entries. See below.
`sitn_id`	Number	Situation ID.
`thread_name`	String	Name of the thread that the entries are from.

The **`entries`** list contains the following information:

Name	Type	Description
`entry_text`	String	Text of the thread entry. Reserved characters are converted to HTML entities, for example, **`&`** is converted to **`&`**.
`user_id`	Number	User ID of the user that created the thread entry.
`time`	Number	Time when the thread entry was created. This is a Unix epoch timestamp in seconds.
`entry_id`	Number	ID of the thread entry.

Examples

The following examples demonstrate typical use of endpoint **`getThreadEntries`**:

Request example

Example cURL request to return the first 10 thread entries on thread "Support" in Situation 358:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getThreadEntries" \
--data-urlencode "sitn_id=358" \
--data-urlencode "thread_name=Support" \
--data-urlencode "start=0" \
--data-urlencode "limit=10"
```

Response example

Example response returning the two thread entries on thread "Support" in Situation 358:

```
{
    "entries":
    [
        {
            "entry_text":"Test Entry",
            "user_id":4,
            "time":1549455051,
            "entry_id":2
        },
        {
            "entry_text":"Test Entry",
            "user_id":4,
            "time":1549455053,
            "entry_id":1
        }
```

```
        ],
        "sitn_id":358,"thread_name":"Support"
    }
```

getThreadEntry

A Graze API GET request that returns a thread entry specified using the thread entry ID. Threads are comments or 'story activity' on Situations.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getThreadEntry`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	No	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`entry_id`	Number	Yes	Thread entry ID.

Response

Endpoint **`getThreadEntry`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`entry_id`	Number	Thread entry unique ID.
`sig_id`	Number	Situation ID.
`thread_id`	String	Thread identifier. Can be either **`support`** or **`end user`**.
`standard_thread`	String	Standard thread.
`status`	Number	Situation status.
`timed_at`	Number	Timestamp of the thread entry.
`uid`	Number	User ID.
`did`	Number	Department ID of the user.
`entry`	String	Text of the thread entry.
`mmid`	Number	Multimedia database reference of embedded resource.

Examples

The following examples demonstrate typical use of endpoint **`getThreadEntry`**:

Request example

Example cURL request to return thread entry with entry ID "1":

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getThreadEntry" \
--data-urlencode "entry_id=1"
```

Response example

Example response returning thread entry with ID "1" on thread "Support":

```
{
    "entry_id": 1,
    "sig_id": 1,
    "thread_id": "Support",
    "standard_thread": "Support",
    "status": 1,
    "timed_at": 1586874842,
    "uid": 3,
    "did": 1,
    "entry": "My thread entry",
    "mmid": -1
}
```

getToolShares

A Graze API GET request that returns the shared access for a specified tool.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getToolShares** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
tool_id	Number	Yes	ID of the tool that you want to retrieve its shared access for.

Response

Endpoint **getToolShares** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
tool_id	Number	ID of the tool you requested to return its shared access for.
domain_ids	Array	An array of all the IDs within the domain that can access the tool. If the domain is global, no domain IDs are returned.
domain	String	Domain that can access the tool. One of: **user**, **team**, **role**, or **global**.

Examples

The following examples demonstrate typical use of endpoint **getToolShares**:

Request example

Example cURL request to retrieve all the domain IDs that have access to tool 15:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/getToolShares" \
-H "Content-Type: application/json; charset=UTF-8" \
-d '{ "tool_id":15 }'
```

Response example

Example response returning that tool ID 15 can be accessed by team ID 3:

```
{
    "tool_id": 15,
    "domain_ids": [3],
    "domain": "team"
}
```

getTopPrcDetails

A Graze API GET request that returns the top most likely causal alerts, based on their Probable Root Cause value, for a specified Situation.

You can select the maximum number of causal alerts to return using a limit value. If not specified, the endpoint only returns the alert with the highest root cause probability.

The entries returned are ordered with the highest root cause probability first, for the specified Situation, irrespective of whether they have been labeled causal or are unlabeled. Alerts marked as symptoms are excluded from the return.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getTopPrcDetails`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Integer	Yes	ID of the Situation you want to retrieve the Probable Root Cause details for.
`limit`	Integer	No	Maximum number of causal or unlabeled alerts to return. Default is 1, if not specified, returning one alert with the highest root cause probability.

Response

Endpoint **`getTopPrcDetails`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description

rc_probability	Number	Root cause probability of the alert.
description	String	Description of the alert.
rc_label	Integer	Label defining whether the alert is causal or unlabeled. Alerts marked as symptoms are excluded from the return. 1 = causal 0 = unlabeled -1 = symptom
alert_id	Integer	Alert ID.

Examples

The following examples demonstrate typical use of endpoint **getTopPrcDetails**:

Request example

Example cURL request to return the top three causal alerts with the highest root cause probability in Situation 145:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getTopPrcDetails" --
data-urlencode 'sitn_id=145' --data-urlencode 'limit=3'
```

Response example

Example response returning the top three causal or unlabeled alerts for Situation ID 145:

```
{
    "alerts":
    [
        {
            "rc_probability":0.9933107459030244,
            "description":"Web Server HTTPD is DOWN",
            "rc_label":1,
            "alert_id":53
        },
        {
            "rc_probability":0.9933092393241993,
            "description":"Web Server HTTPD is DOWN",
            "rc_label":1,
            "alert_id":8
        },
        {
            "rc_probability":0.22480057080448923,
            "description":"Web Server HTTPD is DOWN",
            "rc_label":0,
            "alert_id":39
        }
    ]
}
```

getUserInfo

A Graze API GET request that returns information about a specified user.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getUserInfo** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
user_id	Number	Yes	ID of the the user to return information about.
username	String	Yes	A valid username.

Response

Endpoint **getUserInfo** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
user_id	Number	User ID.
full_name	String	Full name of the user.

Examples

The following examples demonstrate typical use of endpoint **getUserInfo**:

Request example

Example cURL request to return the information associated with user ID 57:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getUserInfo" \
--data-urlencode "user_id=57"
```

Response example

Example response returning the user information related to user ID 57:

```
{ "full_name":"Lonnie Holmes","user_id":57 }
```

getUserRoles

A Graze API GET request that returns the specified user's roles from the database.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getUserRoles** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
user_id	Number	No, if you specify	User ID.

		username.	
username	String	No, if you specify **user_id**.	A valid username.

Response

Endpoint **getUserRoles** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
JSON Object	JSON	An array javascript object containing the role IDs, the role names and the role descriptions assigned to the user.

Examples

The following examples demonstrate typical use of endpoint **getUserRoles**:

Request example

Example cURL request to return the assigned roles for user "bigfish917":

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getUserRoles" \
--data-urlencode "username=bigfish917"
```

Response example

Example response returning the roles assigned to the user:

```
[
    {
        "id" : 2,
        "name" : "Administrator",
        "description" : "Administrator"
    },
    {
        "id" : 4,
        "name" : "Operator",
        "description" : "Operator"
    },
    {
        "id" : 5,
        "name" : "Customer",
        "description" : "Customer"
    }
]
```

getUsers

A Graze API GET request that returns a list of all users in the database.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getUsers** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
limit	Integer	No	Maximum number of results to return. Default is 1,000.

Response

Endpoint **getUsers** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
JSON Object	JSON	A JSON list of all users, displaying the user ID, teams, full name and username of each user.

Examples

The following examples demonstrate typical use of endpoint **getUsers**:

Request example

Example cURL request to return a maximum of three users:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getUsers" \
--data-urlencode "limit=3"
```

Response example

Example response returning a maximum of three users:

```
[
    {
        "uid": 3,
        "teams": ["Cloud DevOps"],
        "fullname": "Administrator",
        "username": "admin"
    },
    {
        "uid": 6,
        "teams": ["Network"],
        "fullname": "Nagios",
        "username": "Nagios"
    },
    {
        "uid": 5,
        "teams": ["Application Support"],
        "fullname": "Webhook",
        "username": "Webhook"
    }
  ]
```

getUserSessionInfo

A Graze API GET request that returns session information for a single user over a period of time.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getUserSessionInfo`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`username`	String	Yes	Name of the user.
`from`	Number	No	Start time of the period you want to retrieve session information for. This is in Unix epoch time in seconds. If empty, returns all session information for the user.
`to`	Number	No	End time of the period you want to retrieve session information for. This is in Unix epoch time in seconds. If empty, returns user records to date.
`start`	Number	No	Starting record from which data should be included. Default is 0, the first record.
`limit`	Number	No	Maximum number of records you want to return. Default is 200.

Response

Endpoint **`getUserSessionInfo`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`sessionId`	Number	ID of the session.
`startTime`	Number	Start time of the session, in Unix epoch time.
`lastAccess`	Number	Last access time within the session, in Unix epoch time.

Examples

The following examples demonstrate typical use of endpoint **`getUserSessionInfo`**:

Request example

Example cURL request to return the session information for user "admin:

```
curl -G -u graze:graze -k
"https://localhost/graze/v1/getUserSessionInfo?username=admin" \
-H "accept: application/json"
```

Response example

Example response returning the session information for user "admin":

```
[
    {
        "sessionId": 1,
        "startTime": 1571665580,
        "lastAccess": 1571665582
    },
    {
        "sessionId": 3,
        "startTime": 1571666307,
        "lastAccess": 1571666760
    }
]
```

getUserTeams

A Graze API GET request that returns the team names and IDs associated with the specified user ID or username.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getUserTeams** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
user_id	Number	No, if you specify **username**.	A valid user ID.
username	String	No if you specify **user_id**.	A valid username.

Response

Endpoint **getUserTeams** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
JSON Object	JSON	A Javascript object containing the user ID and the teams that the user belongs to.

Examples

The following examples demonstrate typical use of endpoint **getUserTeams**:

Request example

Example cURL request to return the teams that user "admin" belongs to.

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getUserTeams" \
--data-urlencode "username=admin"
```

Response example

Example response returning the teams associated with username "admin":

```
[
    {
        "id" : 11,
        "name" : "Cloud DevOps"
    },
    {
        "id" : 12,
        "name" : "Network"
    },
    {
        "id" : 2,
        "name" : "Application Support"
    }
]
```

getWorkflowEngineMoolets

A Graze API GET request that returns a list of Workflow Engine Moolets and the functions available in each. This endpoint returns an empty list if Moogfarmd is not running.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **getWorkflowEngineMoolets** takes the following request argument:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

Endpoint getWorkflowEngineMoolets takes no other arguments because this endpoint returns data on all the Workflow Engine Moolets and the workflows associated with them.

Response

Endpoint **getWorkflowEngineMoolets** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Type	Description
List of JSON Objects	A list of Workflow Engine Moolets and information about them.

Examples

The following examples demonstrate typical use of endpoint **getWorkflowEngineMoolets**:

Request example

Example cURL request to return information on all the workflows in all the Workflow Engine Moolets in Cisco Crosswork Situation Manager:

```
curl -X GET -u graze:graze -k -v
"https://localhost/graze/v1/getWorkflowEngineMoolets"
```

Response example

Example response returning information on all the Workflow Engine Moolets in Cisco Crosswork Situation Manager:


```
[
    {
        "active": true,
        "last_updated": 1567420771,
        "moolet_name": "Alert Workflows",
        "functions": {
            "alertInSituation": {
                "decision": true,
                "validators": null,
                "name": "alertInSituation",
                "description": "Check if the alert is in an active Situation.",
                "arguments": [],
                "actionOnAssociated": true,
                "type": ["alert"]
            },
            "alertNotInSituation": {
                "decision": true,
                "validators": null,
                "name": "alertNotInSituation",
                "description": "Check if the alert is not in an active
Situation.",
                "arguments": [],
                "actionOnAssociated": true,
                "type": ["alert"]
            },
            "between": {
                "decision": true,
                "validators": null,
                "name": "between",
                "description": "Check to see if the trigger falls between two
times, and optionally on specific days.",
                "arguments": [
                    {
                        "name": "from",
                        "validator": {
                            "regex": "^[0-9]{2}:[0-9]{2}:[0-9]{2}$"
                        },
                        "description": "The 'from' time in hh:mm:ss 24hr
format",
                        "type": "string",
                        "required": true
                    },
                    {
                        "name": "to",
                        "validator": {
                            "regex": "^[0-9]{2}:[0-9]{2}:[0-9]{2}$"
                        },
                        "description": "The 'to' time in hh:mm:ss 24hr format",
                        "type": "string",
                        "required": true
                    },
                    {
```

```
                        "name": "days",
                        "description": "The optional list of days in short form
(Mon,Tue,Wed...), for all days use a blank list []",
                        "type": "object",
                        "required": true
                    }
                ],
                "actionOnAssociated": false,
                "type": ["alert","situation"]
            },
            "contains": {
                "decision": true,
                "validators": null,
                "name": "contains",
                "description": "Check whether the specified object field
contains any of the listed values. Define values as an array, for example [ a ]
or [ a, b, c ].",
                "arguments": [
                    {
                        "name": "field",
                        "description": "The name of the object field to check
values in (including custom_info).",
                        "type": "string",
                        "required": true
                    },
                    {
                        "name": "values",
                        "description": "The list of values to check for, any
intersection is valid.",
                        "type": "object",
                        "required": true
                    }
                ],
                "actionOnAssociated": true,
                "type": ["event","alert","situation"]
            },
            "containsAll": {
                "decision": true,
                "validators": null,
                "name": "containsAll",
                "description": "Check whether the specified object field
contains all of the listed values. Define values as an array, for example [ a ]
or [ a, b, c ].",
                "arguments": [
                    {
                        "name": "field",
                        "description": "The name of the object field to check
values in (including custom_info).",
                        "type": "string",
                        "required": true
                    },
                    {
                        "name": "values",
                        "description": "The list of values to check for, all
must be included to be valid.",
                        "type": "object",
                        "required": true
                    }
                ],
                "actionOnAssociated": true,
                "type": ["event","alert","situation"]
```

```
            },
            "doesNotContain": {
                "decision": true,
                "validators": null,
                "name": "doesNotContain",
                "description": "Check whether the specified object field does
not contain any of the listed values. Define values as an array, for example [ a
] or [ a, b, c ].",
                "arguments": [
                    {
                        "name": "field",
                        "description": "The name of the object field to check
values in (including custom_info).",
                        "type": "string",
                        "required": true
                    },
                    {
                        "name": "values",
                        "description": "The list of values to check for, any
intersection will count.",
                        "type": "object",
                        "required": true
                    }
                ],
                "actionOnAssociated": true,
                "type": ["event","alert","situation"]
            }
        },
       "moolet_type": "alert"
    },
    {
        "active": true,
        "last_updated": 1567420777,
        "moolet_name": "Enrichment Workflows",
        "functions": {
            "alertInSituation": {
                "decision": true,
                "validators": null,
                "name": "alertInSituation",
                "description": "Check if the alert is in an active Situation.",
                "arguments": [],
                "actionOnAssociated": true,
                "type": ["alert"]
            },
            "alertNotInSituation": {
                "decision": true,
                "validators": null,
                "name": "alertNotInSituation",
                "description": "Check if the alert is not in an active
Situation.",
                "arguments": [],
                "actionOnAssociated": true,
                "type": ["alert"]
            },
            "between": {
                "decision": true,
                "validators": null,
```

```
                "name": "between",
                "description": "Check to see if the trigger falls between two
times, and optionally on specific days.",
                "arguments": [
                    {
                        "name": "from"
,                        "validator": {
                            "regex": "^[0-9]{2}:[0-9]{2}:[0-9]{2}$"
                        },
                        "description": "The 'from' time in hh:mm:ss 24hr
format",
                        "type": "string",
                        "required": true
                    },
                    {
                        "name": "to",
                        "validator": {
                            "regex": "^[0-9]{2}:[0-9]{2}:[0-9]{2}$"
                        },
                        "description": "The 'to' time in hh:mm:ss 24hr format",
                        "type": "string",
                        "required": true
                    },
                    {
                        "name": "days",
                        "description": "The optional list of days in short form
(Mon,Tue,Wed...), for all days use a blank list []",
                        "type": "object",
                        "required": true
                    }
                ],
                "actionOnAssociated": false,
                "type": ["alert","situation"]
            },
        "moolet_type": "alert"
    },
]
```


getWorkflows

A Graze API GET request that returns workflows for a Workflow Engine Moolet.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`getWorkflows`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`moolet_name`	String	Yes	Name of the Workflow Engine Moolet that you want to return the workflows for.

Response

Endpoint **`getWorkflows`** returns the following response:

Type	Description
HTTP	HTTP status or error code indicating request success or failure. See HTTP status code

Code	definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
id	Integer	Unique ID of the workflow.
moolet_name	String	Name of the Workflow Engine Moolet.
workflow_name	String	Name of the workflow.
description	String	Description of the workflow.
sequence	Integer	Sequence number of the workflow.
active	Boolean	Indicates whether or not the Moolet's associated Workflow Engine is active.
entry_filter	String	An SQL-like filter to determine which events, alerts, or Situations can enter the workflow. If empty, the workflow accepts all events, alerts or Situations.
sweep_up_filter	String	An SQL-like filter to intake any additional alerts or Situations from the database. Not relevant for event workflows.
first_match_only	Boolean	If enabled, alerts and Situations only pass through actions on the first time they enter the Workflow Engine. Not relevant for event workflows.
operations	JSON List	List of properties relating to each operation: Name / Type / Description type / String / Type of operation. Options are: 'action', 'decision' and 'delay'. operation_name / String / Name of the operation. Only relevant for 'action' and 'decision' types. function_name / String / Name of the function. Only relevant for 'action' and 'decision' types. function_args / JSON Object / Arguments for the function. duration / Integer / Length of time before the message goes to the next operation. Only relevant for 'delay' type. reset / Boolean / Determines whether the timer resets after each occurrence. Only relevant for 'delay' type.

Examples

The following examples demonstrate typical use of endpoint **`getWorkflows`**:

Request example

Example cURL request to return workflows associated with the "Alert Workflows" Moolet:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getWorkflows" -H
"Content-Type: application/json; charset=UTF-8" --data-urlencode
"moolet_name=Alert Workflows"
```

Response example

Example response returning information on the workflows associated with the "Alert Workflows" Moolet:

```
[
    {
        "first_match_only": false,
        "sequence": 1,
        "operations": [
        {
            "duration": 0,
            "reset": false,
            "type": "delay"
        },
        {
            "operation_name": "Stop Alert",
            "function_name": "stop",
            "forwarding_behavior": "stop all workflows",
            "type": "action"
        }],
        "moolet_name": "Alert Workflows",
        "workflow_name": "Closed Alerts Filter",
        "entry_filter": "state = 9",
        "active": true,
        "description": "You can optionally use this workflow to prevent closed
alerts from processing.",
        "sweep_up_filter": "((agent = \"Test\") AND (significance = 0)) OR
(severity = 0)",
        "id": 3
    }
]
```

mergeSituations

A Graze API POST request that merges multiple specified Situations. You can configure whether or not the new Situation supersedes the original Situations using the **`supersede_original`** parameter.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`mergeSituations`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`situations`	Array of Numbers	Yes	An array of the Situation IDs you want to merge. Specify using Situation IDs, separating each item with a comma.

supersede_original	Boolean	Yes	Determines whether or not the original merged Situations are superseded by the new Situation.

Response

Endpoint **mergeSituations** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
sitn_id	Number	ID of the new merged Situation.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **mergeSituations**:

Request example

Example cURL request to merge Situations 31, 32, and 33 without superseding the original Situations by the new one:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/mergeSituations?auth_token=c4316d2cac524b96a1e4c787b
68f7e3f&situations=%5B31%2C32%2C33%5D&supersede_original=false"
```

Response example

Example response returning the ID of the new merged Situation:

```
{"sitn_id":30}
```

rateSituation

A Graze API POST request that applies a rating to a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **rateSituation** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sig_id	Long	Yes	ID of the Situation you want to rate.
rating	Integer	Yes	Rating that you want to apply to the Situation. This is equivalent to the number of stars that you can assign to a Situation in the UI. One of:0 = Not yet rated1 = Bad2 = Poor3 = Adequate4 = Good5 = Excellent
comment	String	No	A comment about the rating you are applying to the Situation.

Response

Endpoint **rateSituation** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
rating	Number	Rating number applied to the Situation.
comment	String	Comment applied to the Situation.
sitn_id	Number	ID of the Situation that the rating was applied to.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **rateSituation**:

Request example

Example cURL request to apply a rating of 4 to Situation ID 18 with a comment "Rating 4":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/rateSituation" -H
"Content-Type: application/json; charset=UTF-8" -d '{"sig_id" : 18, "rating" :
"4", "comment" : "Rating 4"}'
```

Response example

Example response returning the rating, comment and Situation ID:

```
{"rating":4,"comment":"Rating 4","sitn_id":18}
```

removeAlertFromSituation

A Graze API POST request that removes a specified alert from a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **removeAlertFromSituation** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
alert_id	Number	Yes	ID of the alert you want to remove from the Situation.
sitn_id	Number	Yes	Situation ID.

Response

Endpoint **removeAlertFromSituation** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

This endpoint does not remove the alert from the Situation if the alert has been archived to the historic database even if the Situation is still in the active database.

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **removeAlertFromSituation**:

Request example

Example cURL request to remove alert 16 from Situation 7:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/removeAlertFromSituation" -H "Content-Type:
application/json; charset=UTF-8" -d '{"alert_id" : 16, "sitn_id" : 7 }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

removeEventsAnalyserPartitionOverrides

A Graze API POST request that removes all the partition overrides in the Events Analyser configuration.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`removeEventsAnalyserPartitionOverrides`** takes the following request argument:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

Endpoint **`removeEventsAnalyserPartitionOverrides`** takes no other arguments because it removes all the partition overrides information in the Events Analyser configuration.

Response

Endpoint **`removeEventsAnalyserPartitionOverrides`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **`removeEventsAnalyserPartitionOverrides`**:

Request example

Example cURL request to remove all the partition overrides in the Events Analyser:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/removeEventsAnalyserPartitionOverrides" -H "Content-
Type: application/json; charset=UTF-8"
```

Response example

A successful request returns the HTTP code 200 and no response text.

removeEventsAnalyserWord

A Graze API POST request that removes a single word from the list of priority words or stop words in the Events Analyser configuration. This endpoint removes the word from the list of priority words or stop words depending on the argument you supply. Use updateEventsAnalyserWords to replace an entire list of priority words or stop words, or addEventsAnalyserWord to add a single word to a list of priority words or stop words.

See updateEventsAnalyserConfig for updating the other fields in the Events Analyser configuration.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`removeEventsAnalyserWord`** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
type	String	Yes	Determines whether the endpoint removes the word from the list of priority words or stop words. Set to **priority_word** to remove the word from the list of priority words. Set to **stop_word** to remove the word from the list of stop words.

Response

Endpoint **removeEventsAnalyserWord** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **removeEventsAnalyserWord**:

Request examples

Example cURL request to remove the word 'fail' from the list of priority words in the Events Analyser configuration:

```
curl -POST -u graze:graze -k -v
"https://localhost/graze/v1/removeEventsAnalyserWord" \
--data-urlencode 'type=priority_word' \
--data-urlencode 'word="fail"'
```

Example cURL request to remove the word 'then' from the list of stop words in the Events Analyser configuration:

```
curl -POST -u graze:graze -k -v
"https://localhost/graze/v1/removeEventsAnalyserWord" \
--data-urlencode 'type=stop_word' \
--data-urlencode 'word="then"'
```

Response example

A successful request returns the HTTP code 200 and no response text.

removeSigCorrelationInfo

A Graze API DELETE request that removes all correlation information related to a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **removeSigCorrelationInfo** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

sitn_id	Number	Yes	Situation ID.
service_name	String	No	Service name.
external_id	String	No	External ID.

Response

Endpoint **removeSigCorrelationInfo** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **removeSigCorrelationInfo**:

Request example

Example cURL request to remove the correlation information from Situation ID 3 for service name "my service 7" and external ID "my resource 7":

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/removeSigCorrelationInfo" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 3, "service_name" : "my
service 7", "external_id" : "my resource 7"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

removeSituationPrimaryTeam

A Graze API POST request that removes the primary team from a Situation. The team remains assigned to the Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **removeSituationPrimaryTeam** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

sitn_id	Number	Yes	ID of the Situation that you want to remove the primary team from.

Response

Endpoint **removeSituationPrimaryTeam** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
sitn_id	Number	ID of the Situation that the primary team has been removed from.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **removeSituationPrimaryTeam**:

Request example

Example cURL request to remove the primary team from Situation 1906:

```
curl -G -u graze:graze -k
"https://localhost/graze/v1/removeSituationPrimaryTeam" --data-urlencode
'sitn_id=1906'
```

Response example

Example response returning the Situation ID that the primary team has been removed from:

```
{
    "sitn_id": 1906
}
```

reorderWorkflows

A Graze API POST request that reorders the sequence of workflows within a Workflow Engine Moolet.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **reorderWorkflows** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
moolet_name	String	Yes	Name of the Workflow Engine Moolet.
workflow_sequence	Array of Integers	Yes	An ordered array of all the workflow IDs within the Workflow Engine Moolet. The position of each workflow ID is its position within the Workflow Engine Moolet.

Response

Endpoint **reorderWorkflows** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **reorderWorkflows**:

Request example

Example cURL request to order the workflows within the "Alert Workflows" Workflow Engine as workflow ID 3 then workflow ID 1:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/reorderWorkflows"
\
-H "Content-Type: application/json; charset=UTF-8" \
--data '{ "moolet_name" : "Alert Workflows","workflow_sequences" : [3,1] }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

resolveAlerts

A Graze API POST request that resolves a list of alerts.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **resolveAlerts** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
alert_ids	Array of Numbers	Yes	List of IDs of the alerts you want to resolve.
thread_entry_comment	String	No	Thread entry comment you want to add to the resolved alerts. HTML and XML tags are stripped from the thread entry text. Reserved characters are converted to HTML entities, for example, **&** is

			converted to **&**.

Response

Endpoint **resolveAlerts** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
status	Boolean	Whether or not the alerts were resolved.
resolved_alerts	Number list	List of IDs of alerts that were resolved.
failed_alerts	Number list	List of IDs of alerts that failed to be resolved.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **resolveAlerts**:

Request example

Example cURL request to set alerts 45, 76, and 352 as resolved with the comment "Resolved'":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/resolveAlerts" -H
"Content-Type: application/json; charset=UTF-8" -d '{"alert_ids" : [45,76,352],
"thread_entry_comment" : "Resolved"}'
```

Response example

Example response showing that alerts 45, 76 and 352 were successfully resolved and no alerts failed:

```
{"status":true,"resolved_alerts":[45,76,352],"failed_alerts":[]}
```

resolveSituation

A Graze API POST request that resolves a specified Situation that is currently open.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **resolveSituation** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.

Response

Endpoint **resolveSituation** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **resolveSituation**:

Request example

Example cURL request to mark Situation ID 5 as resolved:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/resolveSituation"
-H "Content-Type: application/json; charset=UTF-8" -d '{"sitn_id" : 5}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

sendToWorkflow

A Graze API POST request that sends a Moolet Inform message to a workflow in an Inform Workflow Engine.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **sendToWorkflow** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request.

			See the authenticate endpoint for more information.
engine_name	String	Yes	Name of an active Inform Workflow Engine.
workflow_name	String	Yes	Name of an active workflow within the specified Inform Workflow Engine.
sitn_id	Number	No	ID of the Situation you want to send to the workflow.
alert_id	Number	No	ID of the alert you want to send to the workflow.
context	Map	No	Additional context to send with the message. This must be available as an action in the workflow as **getWorkflowContext()**.

Response

Endpoint **sendToWorkflow** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **sendToWorkflow**:

Request examples

Example cURL request to send a message to a Situation Inform Workflow Engine about Situation ID 12:

```
curl -X POST -u graze:graze -k "https://localhost/graze/v1/sendToWorkflow" \
-H "Content-Type: application/json; charset=UTF-8" \
--data '{
    "engine_name" : "Situation Inform Engine",
    "workflow_name": "Workflow name",
    "sitn_id": 12,
    "context": {"hello": "world"}
    }'
```

Example cURL request to send a message to an alert Inform Workflow Engine about alert ID 35:

```
curl -X POST -u graze:graze -k "https://localhost/graze/v1/sendToWorkflow" \
-H "Content-Type: application/json; charset=UTF-8" \
--data '{
    "engine_name" : "Alert Inform Engine",
    "workflow_name": "Workflow name",
    "alert_id": 35,
    "context": {"hello": "world"}
    }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

setAlertAcknowledgeState

A Graze API POST request that acknowledges or unacknowledges the owner of the specified alert ID.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **setAlertAcknowledgeState** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
alert_id	Number	Yes	ID of the alert that you want to acknowledge or not acknowledge.
acknowledged	Number	Yes	The acknowledge state you want to apply to the alert: 0 for unacknowledged, 1 for acknowledged.

Response

Endpoint **setAlertAcknowledgeState** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **setAlertAcknowledgeState**:

Request example

Example cURL request to set the acknowledge state of alert ID 7 to "acknowledged":

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/setAlertAcknowledgeState" -H "Content-Type:
application/json; charset=UTF-8" -d '{"alert_id" : 7, "acknowledged" : 1 }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

setAlertSeverity

A Graze API POST request that sets the severity level of an alert.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **setAlertSeverity** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
alert_id	Number	Yes	Alert ID.
severity	Number	Yes	The severity of the alert as an integer:0 = Clear1 = Indeterminate2 = Warning3 = Minor4 = Major5 = Critical

Response

Endpoint **setAlertSeverity** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **setAlertSeverity**:

Request example

Example cURL request to set the alert with ID 7 as "Critical":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/setAlertSeverity"
-H "Content-Type: application/json; charset=UTF-8" -d '{"alert_id" : 7,
"severity" : 5 }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

setGlobalEntropyThreshold

A Graze API POST request that sets the global default entropy threshold or a manager-specific entropy threshold, either as a value or as a percentage.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`setGlobalEntropyThreshold`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`type`	String	Yes	Either: **`entropy_value`**: Indicates that you want to set a global entropy threshold value. **`percentage_reduction`**: Indicates that you want to set a percentage that you want to reduce the current global entropy threshold by.
`value`	Number	Yes	Entropy threshold. A number between 0.0 and 1.0 for both an entropy value and a percentage.
`name`	String	No	Name of the entropy threshold.
`filter`	String	No	An SQL-like filter of the manager.

Response

Endpoint **`setGlobalEntropyThreshold`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **`setGlobalEntropyThreshold`**:

Request example

Example cURL request to set an entropy threshold as a percentage of 50% for manager `"manager1"`:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/setGlobalEntropyThreshold" -H "Content-Type:
application/json; charset=UTF-8" -d '{ \
"type" : "percentage_reduction", \
"value" : 0.5, \
"name" : "manager1", \
"filter" : "manager='manager1'" \
}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

setPrcLabels

A Graze API POST request that sets the probable root cause (PRC) labels for specified alerts within a Situation. You must specify at least one PRC level and an alert ID for that level.

You can mark alerts as causal, non-causal or unlabeled within a Situation. An alert can have different PRC levels within different Situations.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`setPrcLabels`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.
`causal`	Array of Numbers	No, if you specify **`non_causal`** or **`unlabelled`**.	A list of alert IDs that you want to be marked as causal.
`non_causal`	Array of Numbers	No, if you specify **`causal`** or **`unlabelled`**.	A list of alert IDs that you want to be marked as non-causal.
`unlabelled`	Array of Numbers	No, if you specify **`causal`** or **`non_causal`**.	A list of alert IDs that you want to be marked as unlabeled.

Response

Endpoint **`setPrcLabels`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`setPrcLabels`**:

Request example

Example cURL request to set alert ID 1 as causal, alert IDs 2 and 3 as non-causal, and alert 4 as unlabeled:

```
curl -POST -u graze:graze -k -v "https://localhost/graze/v1/setPrcLabels" --
data-urlencode "sitn_id=1" --data-urlencode "causal=[1]" --data-urlencode
"non_causal=[2,3]" --data-urlencode "unlabelled=[4]"
```

Response example

A successful request returns the HTTP code 200 and no response text.

setResolvingThreadEntry

A Graze API POST request that sets or clears a thread entry in a Situation as a resolving step. Threads are comments or 'story activity' on Situations.

This endpoint returns a Boolean indicating whether the thread entry was successfully set or cleared as a resolving step.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`setResolvingThreadEntry`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`entry_id`	Number	Yes	ID of the thread entry.
`resolving_step`	Boolean	Yes	Whether you are setting or clearing the thread entry as a resolving step.

Response

Endpoint **`setResolvingThreadEntry`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Type	Description
Boolean	Whether or not the thread entry was successfully set or cleared as a resolving step.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`setResolvingThreadEntry`**:

Request example

Example cURL request to set thread entry 28 as a resolving step:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/setResolvingThreadEntry" -H "Content-Type:
application/json; charset=UTF-8" -d '{"entry_id" : 28, "resolving_step" : true}'
```

Response example

Example response returning that the thread entry was successfully set as a resolving step:

```
true
```

setSituationAcknowledgeState

A Graze API POST request that acknowledges or unacknowledges the moderator who has been assigned to a Situation. The Situation must be assigned for this request to be successful.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`setSituationAcknowledgeState`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	Situation ID.
`acknowledged`	Number	Yes	The acknowledge state: 0 = unacknowledged 1 = acknowledged

Response

Endpoint **`setSituationAcknowledgeState`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **setSituationAcknowledgeState**:

Request example

Example cURL request to set the moderator on Situation ID 64 as acknowledged:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/setSituationAcknowledgeState" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 64, "acknowledged" : 1 }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

setSituationDescription

A Graze API POST request that sets the description for a specified Situation.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **setSituationDescription** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.
description	String	Yes	Description for the Situation ID.

Response

Endpoint **setSituationDescription** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **setSituationDescription**:

Request example

Example cURL request to set the description for Situation ID 6 as "This is my description 12345":

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/setSituationDescription" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 6, "description" : "This is my
description 12345"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

setSituationFlags

A Graze API POST request that updates the flags associated with a specified Situation. You can add flags to or remove them from a Situation.

See Situation Flags for more information on Cisco Crosswork Situation Manager Situation flags.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`setSituationFlags`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_ids`	Array of Numbers	Yes	An array of IDs for the Situations you want to update.
`to_add`	Array of Strings	Yes	Flags to be added to those Situations. If this is an empty list, no flags are added to the Situation.
`to_remove`	Array of Strings	Yes	Flags you want to remove from the Situation. If this is an empty list, no flags are removed from the Situation.

Response

Endpoint **`setSituationFlags`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes

Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`setSituationFlags`**:

Request example

Example cURL request to change the flags assigned to a situation. This change can include adding and/or removing flags. If one of the change arguments is left empty, nothing will change for that action.

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/setSituationFlags"
-H "Content-Type: application/json; charset=UTF-8" -d '{"sitn_ids":[ 1 ],
"to_add": [ "NOTIFIED","TICKETED"],"to_remove": [] }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

setSituationPrimaryTeam

A Graze API POST request that sets one of the teams already assigned to a Situation as the primary team.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`setSituationPrimaryTeam`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	ID of the Situation.
`team_id`	Number	No, if you specify **`team_name`**.	ID of the team that you want to make the primary team.
`team_name`	String	No, if you specify **`team_id`**.	Name of the team that you want to make the primary team.

Response

Endpoint **`setSituationPrimaryTeam`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`sitn_id`	Number	ID of the Situation.
`primary_team_id`	Number	ID of the primary team.

primary_team_name	String	Name of the primary team.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **setSituationPrimaryTeam**:

Request example

Example cURL request to set the team "Database Management System" as the primary team on Situation 1906:

```
curl -X POST -u graze:graze -k
"https://localhost/graze/v1/setSituationPrimaryTeam" -H "Content-Type:
application/json; charset=UTF-8" --data '{
    "sitn_id" : 1906,
    "team_name" : "Database Management System"
}'
```

Response example

Example response returning that team "Database Management System" is the primary team on Situation 1906:

```
{
    "sitn_id": 1906,
    "primary_team_id": 12,
    "primary_team_name": "Database Management System"
}
```

setSituationProcesses

A Graze API POST request that applies a list of processes to a specified Situation. Any other processes already associated with the Situation are removed.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **setSituationProcesses** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request.

			See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.
process_list	Array of Strings	Yes	A Javascript array of process names as text strings. If any processes supplied do not exist in the database, the request creates them and assigns them to the Situation.

Response

Endpoint **setSituationProcesses** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **setSituationProcesses**:

Request example

Example cURL request to set the processes for Situation ID as "Knowledge Management" and "90nm Manufacturing":

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/setSituationProcesses" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 7, "process_list" :
["Knowledge Management", "90nm Manufacturing"]}
```

Response example

A successful request returns the HTTP code 200 and no response text.

setSituationServices

A Graze API POST request that applies a list of external services to a specified Situation. Any other services already associated with the Situation are removed.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **setSituationServices** takes the following request arguments:

Name	Type	Required	Description

auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
sitn_id	Number	Yes	Situation ID.
service_list	Array of Strings	Yes	A Javascript array of service names as text strings. If any services supplied do not exist in the database, the request creates them and assigns them to the Situation.

Response

Endpoint **setSituationServices** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **setSituationServices**:

Request example

Example cURL request to [complete]:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/setSituationServices" -H "Content-Type:
application/json; charset=UTF-8" -d '{"sitn_id" : 8, "service_list" :
["Knowledge Management", "90nm Manufacturing"]}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

shareToolAccess

A Graze API POST request that shares access to a tool with other users, teams, or roles, or makes it global so that all users can access it. When a user creates a tool, it is automatically shared globally. You can use this endpoint to restrict its availability and ensure that tools are only available to users who need them. Using this endpoint to share access to a tool overwrites any existing shares.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **shareToolAccess** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
tool_id	Number	Yes	ID of the tool that you want to share access for.
domain	String	Yes	Domain to share access with. One of: **user**, **team**, **role**, or **global**.
domain_ids	Array	Yes/No	An array of one or more IDs within the domain. Optional for the **global** domain.

Response

Endpoint **shareToolAccess** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
domain_ids	Array	An array of the IDs for the domain that you requested to share tool access with.
domain	String	Domain that you requested to share access with. One of: **user**, **team**, **role**, or **global**.
tool_id	Number	ID of the tool you requested to share access with.

Examples

The following examples demonstrate typical use of endpoint **shareToolAccess**:

Request example

Example cURL request to share access of tool ID 15 with team ID 3:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/shareToolAccess" -
H "Content-Type: application/json; charset=UTF-8" -d '{"tool_id":15,
"domain":"team", "domain_ids":[3]}'
```

Response example

Example response returning that the request to share access of tool ID 15 with team ID 3 was successful:

```
{
    "domain_ids": [
        3
    ],
    "domain": "team",
    "tool_id": 15
}
```

/situation/{situationID}/topologies

The **/situation/{situationID}/topologies** endpoint allows you to retrieve the topologies related to the alerts in a specified Situation.

To retrieve the node and link details for a specified Situation and topology see getSituationTopology.

Back to Topologies API Endpoint Reference.

GET

Retrieves the topologies related to a specified Situation.

Path parameters

The GET request takes the following path parameter:

Name	Type	Required	Description
sitn_id	Number	Yes	ID of the Situation for which to retrieve topologies.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	Name of the topology.
causal	Boolean	Flag indicating whether this topology caused the Situation to be created.

Example

The following example demonstrates making a GET request to the **situation/{situationID}/topologies** endpoint.

Request example

Example cURL request for topologies related to the alerts in Situation with ID 12:

```
curl \
https://example.com/api/v1/situation/12/topologies \
-u phil:password123 \
```

Response example

Example response returning the details of two topologies:

```
[
    {
        "name" : "host",
        "causal": true
    },
    {
```

```
            "name" : "location",
            "causal" : false
        }
    ]
```

updateBotRecipe

A Graze API POST request that updates a Cookbook Bot Recipe.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateBotRecipe** takes the following request arguments. You must supply the name of the Bot Recipe plus at least one other argument that you want to change.

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the Recipe that you want to update.
cookbooks	Array of Strings	No	A list of the Cookbooks that this Recipe belongs to.
description	String	No	Description of the Recipe.
alert_threshold	Positive Integer	No	Minimum number of alerts required before Cookbook creates a Situation.
trigger	String	No	A filter that determines the alerts that Cookbook considers for Situation creation. Cookbook includes alerts that match the trigger filter. By default Cookbook only includes alerts with a severity of 'Critical'.
exclusion	String	No	A filter that determines the alerts to exclude from Situation creation. Cookbook ignores alerts that match the exclusion filter. For details on creating a filter, see
seed_alert	String	No	A filter that determines whether to create a Situation from a seed alert. The seed alert must meet both the **trigger**, **exclusion** and **seed_alert** criteria to create a Situation. Cookbook considers subsequent alerts for clustering if they meet the trigger and exclusion filter criteria. Alerts that arrive prior to the seed alert that met the trigger and exclusion filter criteria do not form Situations.
rate	Positive Integer	No	Rate, in number of alerts per second. Cookbook clusters alerts if they arrive at a higher rate than is specified here. Cookbook uses **rate** together with **min_sample_size** and **max_sample_size** to determines whether to cluster alerts into Situations. See Cookbook and Recipe Examples.
min_sample_size	Positive Integer	No	Number of alerts that must arrive before the Cookbook starts to calculate the alert rate. See Cookbook and Recipe Examples. Valid only if **rate** is non-

			zero.Cookbook and Recipe Examples
max_sample_size	Positive Integer	No	Maximum number of alerts that are considered in the alert rate calculation. When more than this number of alerts have arrived, Cookbook discards the oldest alerts and calculates the alert rate based on the number of alerts in the **max_sample_size**. See Cookbook and Recipe Examples for more information. Valid only if **rate** is non-zero.Cookbook and Recipe Examples
cook_for	Positive Integer	No	Minimum time period, in seconds, that the Cookbook Recipe clusters alerts for before it resets and starts a new cluster. See Cookbook and Recipe Examples for more information.Cookbook and Recipe Examples If you set a different **cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for** time inherit the value from the Cookbook.
cook_for_extension	Positive Integer	No	Time period that the Cookbook Recipe can extend clustering alerts for before it resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Cookbook. As Cookbook receives related alerts, it continues to extend the total clustering time until the **max_cook_for** period is reached. Used in conjunction with the **max_cook_for** value, the **cook_for_extension** period helps to ensure that Cookbook continues to cluster alerts together that are related to the same failure. The **cook_for_extension** period only applies to new related alerts; it does not apply to existing alerts that are updated with new events. See Cookbook and Recipe Examples for more information. If you set a different **cook_for_extension** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for_extension** time inherit the value from the Cookbook.
max_cook_for	Positive Integer	No	Maximum time period that the Cookbook Recipe clusters alerts for before it resets and starts a new cluster. It works in conjunction with the **cook_for_extension** time to help ensure that Cookbook continues to cluster alerts together that are related to the same failure. This value is ignored unless the **cook_for_extension** time is specified. See Cookbook and Recipe Examples for more information.Cookbook and Recipe Examples If you set a different **max_cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **max_cook_for** value inherit the value from the Cookbook.

cluster_by	String	No	Determines Cookbook's clustering behavior. Set to an empty string to use the Cookbook **cluster_by** setting. Set to **first_match** so that Cookbook adds alerts to the first cluster over the similarity threshold value. Set to **closest_match** to add alerts to the cluster with the highest similarity greater than the similarity threshold value. This option may be less efficient because Cookbook needs to compare alerts against each cluster in a Recipe. Set to an empty string to use the Cookbook setting. If you set a different **cluster_by** value for a Recipe, it overrides the Cookbook value. Recipes without a **cluster_by** value inherit the value from the Cookbook.
initialize_function	JSON Function Name	No	Default is **initBuckets**.
member_function	JSON Function Name	No	Default is **checkBucket**.
can_start_cluster	JSON Function Name	No	Default is null.
use_in_recipe	JSON Function Name	No	Default is null.
similarity	Double	No	Value between 0 and 1. Default is 0.8.

Response

Endpoint **updateBotRecipe** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateBotRecipe**:

Request example

Example cURL request to update the alert threshold to 4 in Bot Recipe "BotRecipe2":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateBotRecipe" -
H "Content-Type: application/json; charset=UTF-8" -d
'{"name":"BotRecipe2","alert_threshold":4}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateClosedAlert

A Graze API POST request that updates the description and custom info of a closed alert during the grace period. The grace period is when an alert is closed and in the active database, before it is archived to the historic database. If a custom info field already exists, this endpoint replaces the previous value; if the custom info field does not exist, this endpoint adds it.

The **`updateClosedAlert`** endpoint returns an error if the alert is open, or if it is closed and has been archived to the historic database.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`updateClosedAlert`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`alert_id`	Number	Yes	ID of the closed alert that you want to update.
`description`	String	No	New description of the alert.
`custom_info`	JSON Object	No	A JSON object containing the custom info values that you want to update. If the key already exists, the endpoint replaces the existing value. If the key does not exist, the endpoint adds it.

Response

Endpoint **`updateClosedAlert`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	No
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`updateClosedAlert`**:

Request examples

Request example to replace description

Example cURL request to update the description for alert ID 21. The **`description`** value `"new_desc"` replaces the previous value.

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateClosedAlert"
-H "Content-Type: application/json; charset=UTF-8" -d '{
"alert_id" : 21, \
"description": "new_desc" \
}'
```

Request example to replace description and custom info

Example cURL request to update the description and custom info for alert ID 21. The **`description`** value `"new_desc"` replaces the previous value. If the custom info fields **`field1`** and **`field2`** did not exist, the endpoint adds them. If the custom info fields **`field1`** and **`field2`** did previously exist, the endpoint overwrites them with the values `"value1"` and `"value2"`.

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateClosedAlert"
-H "Content-Type: application/json; charset=UTF-8" -d '{ \
"alert_id" : 21, \
"description":"new_desc", \
"custom_info": { \
    "field1": "value1", \
    "field2": "value2" \
    }
}'
```

Request example to update custom info

Example cURL request to update the custom info for alert ID 21.

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateClosedAlert"
-H "Content-Type: application/json; charset=UTF-8" -d '{ \
"alert_id" : 21, \
"custom_info": { \
    "field1": "new_value1", \
    "field3": "value3" \
    } \
}'
```

If the alert custom info contained the following fields before the cURL request:

- **`field1`**: value1
- **`field2`**: value2

After the cURL request, the alert custom info contains the following fields. **`field1`** has been overwritten, **`field2`** is unchanged, and **`field3`** has been added.

- **`field1`**: new_value1
- **`field2`**: value2
- **`field3`**: value3

Response example

A successful request returns the HTTP code 200 and no response text.

updateClosedSituation

A Graze API POST request that updates the description and custom info of a closed Situation during the grace period. The grace period is when a Situation is closed and in the active database, before it is archived to the historic database. If a custom info field already exists, this endpoint replaces the previous value; if a custom info field does not exist, this endpoint adds it.

The **`updateClosedSituation`** endpoint returns an error if the Situation is open, or if it is closed and has been archived to the historic database.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`updateClosedSituation`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`sitn_id`	Number	Yes	ID of the closed Situation that you want to update.
`description`	String	No	New description of the Situation.
`custom_info`	JSON Object	No	A JSON object containing the custom info values that you want to update. If the key already exists, the endpoint replaces the existing value. If the key does not exist, the endpoint adds it.

Response

Endpoint **`updateClosedSituation`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

API update behavior

The behavior of this endpoint depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This endpoint updates the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	No
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of endpoint **`updateClosedSituation`**:

Request examples

Request example to replace description and custom info

Example cURL request to update the description and custom info for Situation ID 555. The **`description`** value `"new_desc"` replaces the previous value. If the custom info fields **`field1`** and **`field2`** did not exist, the endpoint adds them. If the custom info fields **`field1`** and **`field2`** did previously exist, the endpoint overwrites them with the values `"value1"` and `"value2"`.

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/updateClosedSituation" -H "Content-Type:
application/json; charset=UTF-8" -d '{ \
"sitn_id" : 555, \
"description":"new_desc", \
"custom_info": { \
    "field1": "value1", \
    "field2": "value2" \
    } \
}'
```

Request example to update custom info

Example cURL request to update the custom info for Situation ID 555.

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateClosedAlert"
-H "Content-Type: application/json; charset=UTF-8" -d '{ \
"sitn_id" : 555, \
"custom_info": { \
    "field1": "new_value1", \
    "field3": "value3" \
    } \
}'
```

If the Situation custom info contained the following fields before the cURL request:

- **`field1`**: value1
- **`field2`**: value2

After the cURL request, the Situation custom info contains the following fields. **`field1`** has been overwritten, **`field2`** is unchanged, and **`field3`** has been added.

- **`field1`**: new_value1
- **`field2`**: value2
- **`field3`**: value3

Response example

A successful request returns the HTTP code 200 and no response text.

updateCookbook

A Graze API POST request that updates a Cookbook.

If you change a Cookbook, see Cookbook Configuration Changes for information on how these changes affect the clusters that Cookbook creates.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateCookbook** takes the following request arguments. You must supply the name of the Cookbook plus at least one other argument that you want to change.

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the Cookbook that you want to update.
description	String	No	Description of the Cookbook.
process_output_of	Array of Strings	No	Defines the source of the alerts that Cookbook processes. You can specify none, one or more Moolets. Typically Cookbook processes the output of its direct upstream neighbor in the processing chain. Usually this is "Alert Workflows" which are the output from the Alert Workflow Engine.si
cluster_by	String	No	Determines Cookbook's clustering behavior. Set to **first_match** so that Cookbook adds alerts to the first cluster over the similarity threshold value. Set to **closest_match** to add alerts to the cluster with the highest similarity greater than the similarity threshold value. This option may be less efficient because Cookbook needs to compare alerts against each cluster in a Recipe. If you set a different **cluster_by** value for a Recipe, it overrides the Cookbook value. Recipes without a **cluster_by** value inherit the value from the Cookbook.
entropy_threshold	Number	No	Minimum entropy value an alert must have in order for Cookbook to consider it for clustering it into a Situation. Cookbook does not include any alerts with an entropy value below the threshold in Situations.
threshold_type	String	No	Type of entropy threshold you want Cookbook to use. One of:**global**: Use the global entropy threshold. This is a single entropy threshold that Cookbook applies to all alerts to eliminate noisy alerts with a lower entropy value.**manager**: Use entropy thresholds set up for individual managers. If the manager for an alert has an entropy threshold set, Cookbook uses this value to eliminate noisy alerts with a lower entropy value. If an alert's manager does not have an entropy threshold, Cookbook uses the global entropy threshold to filter out alerts.**explicit_value**: Use the value set in **entropy_threshold** to eliminate noisy alerts with a lower entropy value.**none**: Do not use entropy thresholds. Cookbook will not filter out any alerts

			based on their entropy value.If you do not specify an entropy threshold, the default is global. The default global entropy threshold is 0. This means that unless you actively set up a global threshold, Cookbook will not filter out any alerts based on entropy values.See Configure Entropy Thresholds for more information on setting global and manager-specific entropy thresholds.
cook_for	Integer	No	Minimum time period, in seconds, that Cookbook clusters alerts for before the Recipe resets and starts a new cluster. See Cookbook and Recipe Examples If you set a different **cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for** time inherit the value from the Cookbook.
cook_for_extension	Integer	No	Time period that Cookbook can extend clustering alerts for before the Recipe resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Cookbook. As Cookbook receives related alerts, it continues to extend the total clustering time until the **max_cook_for** period is reached. Used in conjunction with the **max_cook_for** value, the **cook_for_extension** period helps to ensure that Cookbook continues to cluster alerts together that are related to the same failure. The **cook_for_extension** period only applies to new related alerts; it does not apply to existing alerts that are updated with new events. See Cookbook and Recipe Examples If you set a different **cook_for_extension** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for_extension** time inherit the value from the Cookbook.
max_cook_for	Integer	No	Maximum time period that Cookbook clusters alerts for before the Recipe resets and starts a new cluster. It works in conjunction with the **cook_for_extension** time to help ensure that Cookbook continues to cluster alerts together that are related to the same failure. This value is ignored unless the **cook_for_extension** time is specified. If **cook_for_extension** is set and this value is not set, the default is three times the **cook_for** value. See Cookbook and Recipe Examples If you set a different **max_cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **max_cook_for** value inherit the value from the Cookbook.
scale_by_severity	Boolean	No	Determines whether Cookbook ignores alerts with a severity of 0 (Clear). Set to **true** if you want Cookbook to ignore alerts with a severity of 0 (Clear). Set to **false** if you want Cookbook to

			include alerts with a severity of 0 (Clear).
first_recipe_match_only	Boolean	No	Defines whether Cookbook treats Recipes in priority order. If set to **true**, Cookbook adds an alert to a cluster created by the highest priority Recipe that meets the clustering criteria. The priority order is defined by the order of the Recipes in the **recipes** list. If set to **false**, Cookbook adds an alert to clusters in all the Recipes that meet the clustering criteria.
recipes	Array of Strings	No	A list of the Recipes in this Cookbook. You must supply at least one Recipe. If you set **first_recipe_match_only** to **first_match**, Cookbook uses the order of the Recipes in this list to determine their priority. The first Recipe has the highest priority.
run_on_startup	Boolean	No	Whether Cookbook should start when Moogfarmd starts.
moobot	String	No	The Moobot you want Cookbook to use if there are any Bot Recipes. See Recipe Types for more information.

Response

Endpoint **updateCookbook** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

This endpoint returns an error code if the values of **entropy_threshold** and **threshold_type** are inconsistent. For example, if the **entropy_threshold** is set to 0.4 and **threshold_type** is set to global.

Examples

The following examples demonstrate typical use of endpoint **updateCookbook**:

Request examples

Example cURL request to update the **run_on_startup** and cook for auto-extension arguments for Cookbook 'GrazeCookBook1':

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateCookbook" -H
"Content-Type: application/json; charset=UTF-8" -d \
'{ \
"name" : "GrazeCookBook1", \
"run_on_startup":true, \
"cook_for_extension":7200, \
"max_cook_for":14400 \
}'
```

Example cURL request to update Cookbook 'Default Cookbook' to use entropy thresholds set up for individual managers:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateCookbook" -H
"Content-Type: application/json; charset=UTF-8" -d \
'{ \
"name" : "Default Cookbook", \
"threshold_type": "manager" \
}'
```

Example cURL request to update Cookbook 'GrazeCookbook1' to use an explicit entropy threshold for this Cookbook of 0.15:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateCookbook" -H
"Content-Type: application/json; charset=UTF-8" -d \
'{ \
"name" : "Default Cookbook", \
"entropy_threshold": 0.15, \
"threshold_type": "explicit_value" \
}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateDefaultMergeGroup

A Graze API POST request that updates the default merge group in Cisco Crosswork Situation Manager.

Clustering algorithms, such as Cookbook and Tempus, use the default values in the default merge group unless you have set up custom merge groups with different values to merge Situations from these clustering algorithms. You can set up merge groups using the UI (see Merge Groups for details) or using the Graze API endpoint addMergeGroup.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateDefaultMergeGroup** takes the following request arguments:

Name	Type	Required	Description
alert_threshold	Integer	No	Minimum number of alerts that must be present in a cluster before it can become a Situation in the merge group. Must be greater than or equal to 1. Default value is 1.
situation_similarity_limit	Floating Point	No	Percentage of alerts two Situations must share before they are merged. A value between 0 and 1. Default value is 0.7.

Response

Endpoint **updateDefaultMergeGroup** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateDefaultMergeGroup**:

Request example

Example cURL request to set the default merge group's **alert_threshold** to 2:

```
curl -X POST
-u graze:graze -
k -v "https://example.com/graze/v1/updateDefaultMergeGroup"
-H "Content-Type: application/json; charset=UTF-8"
-d '{
    "alert_threshold":2
    }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateEventsAnalyserConfig

A Graze API POST request that updates the Events Analyser configuration.

You cannot use this endpoint to update the lists of priority words and stop words in the Events Analyser configuration. Use updateEventsAnalyserWords to replace an existing list of priority words or stop words. Use addEventsAnalyserWord to add a single word to a list of priority words or stop words, or removeEventsAnalyserWord to remove a single word.

If you use partitions in the entropy calculations, use updateEventsAnalyserPartitionOverrides to update the Events Analyser configuration with any partition overrides you want to implement.

Back to Graze API EndPoint Reference.

Request arguments

The **updateEventsAnalyserConfig** endpoints accepts the following request arguments. Authenticate the endpoint and provide at least one of the following arguments. The endpoint only updates the properties provided.

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
mask	JSON Object	No	Defines which token types the Events Analyser includes or excludes from its entropy calculation. If a token type is set to **false**, the entropy calculation includes it. If it is set to **true**, the entropy calculation excludes the token type. Masking token types, such as dates or numbers, ensures that tokens are not given a higher entropy value than they should have because of unique numbers or dates. The mask argument contains the following options:**path**: Set to **true** to exclude file paths from the entropy calculation.**ip_address**: Set to **true** to exclude IP addresses from the entropy calculation.**mac_address**: Set to **true** to exclude MAC addresses from the entropy calculation.**url**: Set to **true** to exclude URLs from the entropy calculation.**email**: Set to **true** to exclude email addresses from the entropy calculation.**date_time**: Set to

			true to exclude date and time values from the entropy calculation.**number**: Set to **true** to exclude ordinary numbers from the entropy calculation.**hex**: Set to **true** to exclude hex numbers from the entropy calculation.**oid**: Set to **true** to exclude object identifiers from the entropy calculation.**guid**: Set to **true** to exclude globally unique identifiers, also know as universally unique identifiers (UUIDs), from the entropy calculation.**word**: Set to **true** to exclude words from the entropy calculation. Default is: **{** **"path" : false,** **"ip_address" : false,** **"mac_address" : false,** **"url" : false,** **"email" : false,** **"date_time" : true,** **"number" : true,** **"hex" : false,** **"oid" : false,** **"guid" : false,** **"word" : false** **}**
stop_words	Boolean	No	Indicates whether or not the Events Analyser uses stop words. Stop words are small common words such as 'about', 'at', or 'the'. The Events Analyser automatically excludes stop words from its entropy calculation. Set to **true** to use stop words. Set to **false** if you do not want to use stop words. Default is **true**.
stop_word_length	Number	No	Maximum length of words that are automatically excluded by the Events Analyser from its entropy calculation. For example, a value of **3** means the Events Analyser excludes any words of three or less characters. Default is **0** meaning that no words are excluded from its entropy calculation.
priority_words	Boolean	No	Indicates whether or not the Events Analyser uses priority words. The Events Analyser automatically gives alerts containing any priority words an entropy value of 1. Set to **true** to use priority words. Set to **false** if you do not want to use priority words. Default is **false**.
partition_by	String	No	If you want the Events Analyser to partition the data, enter the property that you want to partition by, for example, **source**. Default is **NULL** so the Events Analyser does not use partitioning. If you want to use partitioning, you must enter any relevant information in **partition_overrides** below.
fields	Array of Strings	No	Properties in each event that contribute to the entropy value calculation. Default is **["description"]**. Cisco recommends providing a single field only.
casefold	Boolean	No	Indicates whether the Events Analyser should consider tokens that differ only by case in its entropy calculation. Set

			to **true** to consider tokens in a different case as the same. Set to **false** to consider tokens in a different case as different. Default is **true**.
stemming	Boolean	No	Indicates whether the Events Analyser considers words with the same word stem as the same word in entropy calculations. For example, should the Events Analyser consider 'fail', 'failed' and 'failing' as the same word. Set to **true** to consider words with the same word stem as the same. Set to **false** to consider consider words with the same word stem as different. Default is **false**.
stemming_language	String	No	Language used in the events. Default is **english**.

Response

Endpoint **updateEventsAnalyserConfig** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateEventsAnalyserConfig**:

Request examples

Example cURL request to enable priority words in the Events Analyser. Use updateEventsAnalyserWords to add a list of priority words to the Events Analyser configuration.

```
curl -POST -u graze:graze -k -v
"https://localhost/graze/v1/updateEventsAnalyserConfig" \
--data-urlencode 'priority_words=true'
```

Example cURL request to enable partitioning in the Events Analyser:

```
curl -POST -u graze:graze -k -v
"https://localhost/graze/v1/updateEventsAnalyserConfig" \
--data-urlencode 'partition_by=source'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateEventsAnalyserPartitionOverrides

A Graze API POST request that updates the Events Analyser with the supplied partition overrides information. This endpoint overwrites any existing partition overrides.

If you use partitions in the Events Analyser configuration, the endpoint enables you to specify overrides for specific partitions. These settings override the default configuration you have specified in the arguments in the endpoint updateEventsAnalyserConfig or in the Cisco Crosswork Situation Manager UI. For example, the default Events Analyser configuration may not use priority words but for one partition, London, you might want to enable priority words and set the priority word list to 'NEW_YORK'

and 'LONDON'. If a partition does not have any overrides, or a property is not set for a partition, the Events Analyser uses the values in the default configuration.

Use updateEventsAnalyserConfig to set the **partition_by** parameter to enable the Events Analyser to calculate entropy by partitions.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateEventsAnalyserPartitionOverrides** takes the following request arguments. If an argument is empty or set to null it is set to null in the database.

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
partition_overrides	JSON Object	Yes	A JSON object containing all the partition override information that you want to create.

The **partition_overrides** object has the following format. For any partitions, define the arguments where you want to override the default Events Analyser configuration.

Name	Type	Required	Description
mask	JSON Object	No	Defines which token types the Events Analyser includes or excludes from its entropy calculation for this partition. If a token type is set to **false**, the entropy calculation includes it. If it is set to **true**, the entropy calculation excludes the token type. Masking token types, such as dates or numbers, ensures that tokens are not given a higher entropy value than they should have because of unique numbers or dates. The mask argument contains the following options:**path**: Set to **true** to exclude file paths from the entropy calculation.**ip_address**: Set to **true** to exclude IP addresses from the entropy calculation.**mac_address**: Set to **true** to exclude MAC addresses from the entropy calculation.**url**: Set to **true** to exclude URLs from the entropy calculation.**email**: Set to **true** to exclude email addresses from the entropy calculation.**date_time**: Set to **true** to exclude date and time values from the entropy calculation.**number**: Set to **true** to exclude ordinary numbers from the entropy calculation.**hex**: Set to **true** to exclude hex numbers from the entropy calculation.**oid**: Set to **true** to exclude object identifiers from the entropy calculation.**guid**: Set to **true** to exclude globally unique identifiers, also know as universally unique identifiers (UUIDs), from the entropy calculation.**word**: Set to **true** to exclude words from the entropy calculation.
stop_words	Boolean	No	Indicates whether or not the Events Analyser uses stop words for this partition. Stop words are small common words such as 'about', 'at', or 'the'. The Events Analyser automatically excludes stop words from its entropy calculation. Set to **true** to use stop words. Set to **false**

			if you do not want to use stop words.
stop_words_list	JSON Array of Strings	No	List of stop words that you want the Events Analyser to ignore in its entropy calculation for this partition.
stop_word_length	Number	No	Maximum length of words that are automatically excluded by the Events Analyser from its entropy calculation for this partition. For example, a value of **3** means the Events Analyser excludes any words of three or less characters.
priority_words	Boolean	No	Indicates whether or not the Events Analyser uses priority words in its entropy calculation for this partition. The Events Analyser automatically gives alerts containing any priority words an entropy value of 1. Set to **true** to use priority words. Set to **false** if you do not want to use priority words.
priority_words_list	JSON Array of Strings	No	List of priority words that you want the Events Analyser to automatically assign an entropy value of 1 in its entropy calculation for this partition.
fields	JSON Array of Strings	No	Properties in each event that contribute to the entropy calculation for this partition. Cisco recommends specifying a single field only.
casefold	JSON Object	No	Indicates whether the Events Analyser should consider tokens that differ only by case in its entropy calculation for this partition. Set to **true** to consider tokens in a different case as the same. Set to **false** to consider tokens in a different case as different.
stemming	Boolean	No	Indicates whether the Events Analyser considers words with the same word stem as the same word in its entropy calculation for this partition. For example, should the Events Analyser consider 'fail', 'failed' and 'failing' as the same word. Set to **true** to consider words with the same word stem as the same. Set to **false** to consider words with the same word stem as different.
stemming_language	String	No	Language used in the events.

Response

Endpoint **updateEventsAnalyserPartitionOverrides** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateEventsAnalyserPartitionOverrides**:

Request example

Example cURL request to update the Events Analyser with partition overrides for two partitions, 'NEW_YORK' and 'LONDON':

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/updateEventsAnalyserPartitionOverrides" -H "Content-
Type: application/json; charset=UTF-8" -d \
'{ \
    "partition_overrides": "{ \
        "NEW_YORK": { \
            "fields": ["description"], \
            "casefold": true, \
            "stop_words": false, \
            "priority_words": false, \
            "stop_word_length": 3 \
        }, \
        "LONDON": { \
            "mask": { \
                "date_time": false, \
                "ip_address": true \
            }, \
            "stemming": true, \
            "stop_words": true, \
            "priority_words": true, \
            "stop_words_list":
["france","germany","italy","peru","india","japan","korea"], \
            "stop_word_length": 1, \
            "priority_words_list": ["reboot","shutdown"] \
        } \
    }" \
}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateEventsAnalyserWords

A Graze API POST request that replaces an existing list of priority words or stop words in the Events Analyser configuration. This endpoint replaces the list of priority words or stop words depending on the argument you supply. Use addEventsAnalyserWord to add a single word to an existing list of priority words or stop words, or use removeEventsAnalyserWord to remove a single word.

See updateEventsAnalyserConfig for updating the other fields in the Events Analyser configuration.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateEventsAnalyserWords** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
type	String	Yes	Determines whether the endpoint replaces the list of priority words or stop words. Set to **priority_word** to replace the list of priority

			words. Set to **stop_word** to replace the list of stop words.
words	Array of Strings	Yes	List of priority words or stop words that you want to replace the existing list.

Response

Endpoint **updateEventsAnalyserWords** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateEventsAnalyserWords**:

Request examples

Example cURL request to replace the existing list of priority words with the list provided:

```
curl -POST -u graze:graze -k -v
"https://localhost/graze/v1/updateEventsAnalyserWords" \
--data-urlencode 'type=priority_word' \
--data-urlencode 'words=["fail", "down", "loss", "low"]'
```

Example cURL request to replace the existing list of stop words with the list provided:

```
curl -POST -u graze:graze -k -v
"https://localhost/graze/v1/updateEventsAnalyserWords" \
--data-urlencode 'type=stop_word' \
--data-urlencode 'words=["the", "and","an","if","at","on"]'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateMaintenanceWindow

A Graze API POST request that updates an existing maintenance window.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateMaintenanceWindow** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
window_id	String	Yes	ID of the maintenance window you want to update.
name	String	No	Name of the maintenance window.
description	String	No	Description of the maintenance window.

filter	String	No	SQL-like filter that alerts must match to be included in the maintenance window.
start_date_time	Unix epoch time in seconds (Number)	No	Start time of the maintenance window. This must be in Unix epoch time in seconds and may be up to 5 years in the future.
duration	Seconds (Number)	No	Duration of the maintenance window in seconds. The minimum duration is 1 second and the maximum is 157784630 seconds (5 years).
forward_alerts	Boolean	No	Determines whether or not alerts should be forwarded to the next Moolet in the processing chain.
recurring_period	Number	No	Whether or not this is a recurring maintenance window. Set this to:**1** for a recurring maintenance window.**0** for a one-time maintenance window.If not specified, default is **0**. If you set this property to **1**, you must specify **recurring_period_units**.
recurring_period_units	Number	No	Specifies the recurring period of the maintenance window, in days, weeks or months. Valid values are:**2** = daily**3** = weekly**4** = monthlyDefault is **0** if **recurring_period** is set to **0**.
timezone	String	No	Time zone that you want the maintenance window to be in. You can only change the time zone if the maintenance window is inactive when you make the request. The time zone must be a valid entry in the IANA Time Zone Database. When scheduling recurring maintenance windows, Cisco Crosswork Situation Manager takes into account any daylight savings time changes for the time zone.

Response

Endpoint **updateMaintenanceWindow** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Type	Description
Array	An array of objects containing details of the returned maintenance windows.

Examples

The following examples demonstrate typical use of endpoint **updateMaintenanceWindow**:

Request examples

Example cURL request to update a number of parameters in the existing maintenance window ID 351:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/updateMaintenanceWindow" -H "Content-Type:
```

```
application/json; charset=UTF-8" -d '{"window_id":351, "name":"Updated name",
"description":"Updated Description", "filter":"source = \"server1\"",
"start_date_time":1546433400, "duration":3600, "forward_alerts":false,
"recurring_period":1, "recurring_period_units":3}'
```

Example cURL request to update the existing maintenance window ID 27 so that it will not occur again:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/updateMaintenanceWindow" -H "Content-Type:
application/json; charset=UTF-8" -d '{"window_id":27, "recurring_period":0,
"recurring_period_units":0}'
```

Example cURL request to update the existing maintenance window ID 144 to be in time zone "Europe/London":

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/updateMaintenanceWindow" -H "Content-Type:
application/json; charset=UTF-8" -d '{"window_id":144, "timezone" :
"Europe/London"}'
```

Response example

Example successful response returning details of the updated maintenance window:

```
{
  "del_flag": false,
  "forward_alerts": false,
  "last_updated": 1574076632,
  "timezone": "Europe/London",
  "description": "Updated Description",
  "recurring_period_units": 3,
  "filter": "source IN (\"server4\", \"server5\")",
  "duration": 3600,
  "recurring_period": 1,
  "name": "Updated name",
  "updated_by": 4,
  "window_id": 144,
  "start_date_time": 1674076188
}
```

updateMergeGroup

A Graze API POST request that updates a custom merge group in Cisco Crosswork Situation Manager.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`updateMergeGroup`** takes the following request arguments:

Name	Type	Required	Description
`name`	String	Yes, along with at least one other argument.	The custom merge group's name.
`moolets`	Array of	No	List of clustering algorithm Moolets to

	Strings		include in the custom merge group.
alert_threshold	Integer	No	Minimum number of alerts that must be present in a cluster before it can become a Situation. Must be greater than or equal to 1. Enter **null** if you want the custom merge group to use the default merge group value. Default merge group value is 2.
situation_similarity_limit	Floating Point	No	Percentage of alerts that two Situations in this merge group must share before they are merged. A value between 0 and 1. Enter **null** if you want the merge group to use the default merge group value. Default merge group value is 0.7.

Response

Endpoint **updateMergeGroup** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateMergeGroup**:

Request example

Example cURL request to update a custom merge group's **situation_similarity_limit**:

```
curl -X POST
-u graze:graze -
k -v "https://example.com/graze/v1/updateMergeGroup"
-H "Content-Type: application/json; charset=UTF-8"
-d '{
    "name":"Merge Group 1",
    "situation_similarity_limit":0.6
    }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateSecurityRealm

A Graze API POST request that updates an existing SAML security realm in the database.

Warning

Warn any users who are logged into Cisco Crosswork Situation Manager using the default realm before using this request. The system may log out users when the updated realm becomes active.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateSecurityRealm** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the security realm.
type	String	Yes	Security realm type. This must be **"SAML2"**.
active	Boolean	Yes	Determines whether the new realm is active or not.
configuration	JSON Object	Yes	JSON object containing the realm configuration. You must include all mandatory configuration properties; otherwise the request returns an error. For information on the configuration properties, see Security Configuration Reference.

Response

Endpoint **updateSecurityRealm** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateSecurityRealm**:

Request example

Example cURL command to update a SAML realm with a new X509 certificate:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/updateSecurityRealm" -d '{
    "name":"mySamlRealm",
    "configuration":  {
        "idpMetadata":"<?xml version=\"1.0\" encoding=\"UTF-
8\"?>\r\n<EntitiesDescriptor Name=\"urn:keycloak\"
xmlns=\"urn:oasis:names:tc:SAML:2.0:metadata\"\r\nxmlns:dsig=\"http:\/\/www.w3.o
rg\/2000\/09\/xmldsig#\">\r\n<EntityDescriptor
entityID=\"http:\/\/moogsaml:18080\/auth\/realms\/master\">\r\n<IDPSSODescriptor
WantAuthnRequestsSigned=\"true\"\r\nprotocolSupportEnumeration=\"urn:oasis:names
:tc:SAML:2.0:protocol\">\r\n<KeyDescriptor
use=\"signing\">\r\n<dsig:KeyInfo>\r\n<dsig:KeyName>l8ddhI8SroeNnlq0TkTxIj2VI-
0bvr2QfG_o32jWeKI<\/dsig:KeyName>\r\n<dsig:X509Data>\r\n<dsig:X509Certificate>MI
ICmzCCAYMCBgFk8A9vMjANBgkqhkiG9w0BAQsFADARMQ8wDQYDVQQDDAZtYXN0ZXIwHhcNMTgwNzMxMT
ExNjQwWhcNMjgwNzMxMTExODIwWjARMQ8wDQYDVQQDDAZtYXN0ZXIwggEiMA0GCSqGSIb3DQEBAQUAA4
IBDwAwggEKAoIBAQCOliZ3dBu696slYduAb1BMuvR1bMdTKVBMICWaEEcS8Rzw8gWthPQpw2e202LjOe
u4VkTVmEEAUa2IrLS4QpYgyhOuzapcIGF4kB0ARebalWa7C9od9%2BeTqWgvXPrDOkzp7g%2B%2Ba5yv
tKxE3ieUORPpACvLWcbkMwyb%2Be5V8%2Bz8n4263Uol8srSaxLsm\/oTozJNwbG%2BbzV8JQHU3xFV5
nFbyNySvc%2B\/B7tDFZuJC5BMu6bwi\/rPqp5OMcuB1W%2BxCcX7IYPphnBjRWNyQJD3gRCkjrujISk
TEcqpZEjR79isbofQaPDi5TSjglPD5rr0OWMVqv91a1\/pVN2y0y%2BRlT8HAgMBAAEwDQYJKoZIhvcN
AQELBQADggEBAAgRhWYKESVsTRAUVYzHYptd3\/eX47%2BTVXhjPO0ORLUJbHtfhgohtyejd6ohazkcS
gMy6%2BwaeVojqq4Q\/tzCOW2EAqO9QOQdaBWOPxDXhJ9TGQJE2E28SS2Gg6paAMfRmtA7c6xXii%2BY
fLo3PG1SSc\/sGe4KIPKflkqqDEqEeaY1olPZU2bLnpMSIui2nK1crE2%2Bt9apLWAGosah6scMGZ9vT
rtOVrNuhB2LuU3cvRQWrUBaQuXQsBV7Q6a8lkrrZ6rjAIbO4vcEL4yjQpnA%2BhetuhBlGPQj6ntuhdn
moKmWYY97wk8eXwblhQxg8GUyfqabfOAKwiGAklxgkexm20M=<\/dsig:X509Certificate>\r\n<\/
dsig:X509Data>\r\n<\/dsig:KeyInfo>\r\n<\/KeyDescriptor>\r\n\r\n<SingleLogoutServ
```

```
ice\r\nBinding=\"urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST\"\r\nLocation=\"http:\/\/moogsaml:18080\/auth\/realms\/master\/protocol\/sa
ml\"
\/>\r\n<SingleLogoutService\r\nBinding=\"urn:oasis:names:tc:SAML:2.0:bindings:HT
TP-
Redirect\"\r\nLocation=\"http:\/\/moogsaml:18080\/auth\/realms\/master\/protocol
\/saml\" \/>\r\n<NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-
format:persistent<\/NameIDFormat>\r\n<NameIDFormat>urn:oasis:names:tc:SAML:2.0:n
ameid-
format:transient<\/NameIDFormat>\r\n<NameIDFormat>urn:oasis:names:tc:SAML:1.1:na
meid-
format:unspecified<\/NameIDFormat>\r\n<NameIDFormat>urn:oasis:names:tc:SAML:1.1:
nameid-format:emailAddress<\/NameIDFormat>\r\n<SingleSignOnService
Binding=\"urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
POST\"\r\nLocation=\"http:\/\/moogsaml:18080\/auth\/realms\/master\/protocol\/sa
ml\"
\/>\r\n<SingleSignOnService\r\nBinding=\"urn:oasis:names:tc:SAML:2.0:bindings:HT
TP-
Redirect\"\r\nLocation=\"http:\/\/moogsaml:18080\/auth\/realms\/master\/protocol
\/saml\"
\/>\r\n<SingleSignOnService\r\nBinding=\"urn:oasis:names:tc:SAML:2.0:bindings:SO
AP\"\r\nLocation=\"http:\/\/moogsaml:18080\/auth\/realms\/master\/protocol\/saml
\"
\/>\r\n<\/IDPSSODescriptor>\r\n<\/EntityDescriptor>\r\n<\/EntitiesDescriptor>",
         "defaultRoles":["Operator"],
         "defaultTeams":["Cloud DevOps"],
         "existingUserMappingField":"username",
         "username":"$username",
         "fullname":"$firstname $lastname",
         "maximumAuthenticationLifetime":60
     }
}'
```

cURL command to deactivate an active SAML realm:

```
curl -X POST -u graze:graze -k -v
"https://localhost/graze/v1/updateSecurityRealm" \
-d "name:mySamlRealm" \
-d "active:false"
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateTeam

A Graze API POST request that updates an existing team.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateTeam** takes the following request arguments:

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
team_id	Number	Yes	Team ID.
name	String	No	Team name. Exclude this attribute to leave Cisco Crosswork

			Situation Manager as it is.
alert_filter	String	No	An SQL-like filter that alerts must match to be assigned to the team. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
services	Array of Strings or Numbers	No	List of the team service names or IDs. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
sig_filter	String	No	An SQL-like filter that Situations must match to be assigned to the team. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
landing_page	String	No	Default landing page for the team. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
active	Boolean	No	Set to **true** if the team is active; set to **false** if the team is inactive. Default is **true**. Exclude this attribute to leave Cisco Crosswork Situation Manager as is.
description	String	No	Team description. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
users	Array of Strings or Numbers	No	List of users in the team, either IDs or usernames. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.

Response

Endpoint **updateTeam** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateTeam**:

Request example

Example cURL request to update the information for team ID 16:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateTeam" \
-H "Content-Type: application/json; charset=UTF-8" \
-d '{"team_id" : 16, "name" : "my team name RENAMED", "active" : true,
"description" : "The team description", "users" : []}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateTempus

A Graze API POST request that updates an existing Tempus Moolet.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateTempus** takes the following request arguments. You must supply the name of the Tempus algorithm plus at least one other argument that you want to change.

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the Tempus algorithm. Must be unique.
description	String	No	Description of the Situations Tempus generates.
entropy_threshold	Number	No	Minimum entropy value for an alert to be clustered into a Situation. Tempus does not cluster any alerts with an entropy value below the threshold into Situations.
threshold_type	String	No	Type of entropy threshold you want Tempus to use. One of:**global**: Use the global entropy threshold. This is a single entropy threshold that Tempus applies to all alerts to eliminate noisy alerts with a lower entropy value.**manager**: Use entropy thresholds set up for individual managers. Tempus uses this value to eliminate noisy alerts with a lower entropy value. If an alert's manager does not have an entropy threshold, Tempus uses the global entropy threshold to filter out alerts.**explicit_value**: Use the value set in **entropy_threshold** to eliminate noisy alerts with a lower entropy value.**none**: Do not use entropy thresholds. Tempus will not filter out any alerts based on their entropy value.If you do not specify an entropy threshold, the default is global. The default global entropy threshold is 0. This means that unless you actively set up a global threshold, Tempus will not filter out any alerts based on entropy values.See Configure Entropy Thresholds for more information on setting global and manager-specific entropy thresholds.
execution_interval	Number	No	Executes Tempus after a defined number of seconds.
window_size	Number	No	Determines the length of time when Tempus analyzes alerts and clusters them into a Situation each time it runs.
bucket_size	Number	No	Determines the time span of each bucket in which alerts are captured. Default bucket size is 5

			seconds.
minimum_arrival_similarity	Number	No	How similar alerts must be to be considered for clustering.
alert_threshold	Number	No	Minimum number of alerts that match the clustering criteria before the Tempus algorithm creates a Situation. When Tempus determines the number of alerts required to create a Situation, it compares the alert threshold values in Tempus and in the merge group that Tempus belongs to, and it uses the higher value. If you are using the default merge group which has an alert threshold of 2, Tempus will never create a Situation containing a single alert. If you want Cisco Crosswork Situation Manager to create Situations with a single alert, consider changing the alert threshold in the default merge group to 1 or creating custom merge groups. See Merge Groups for more information on updating the default merge group and setting up custom merge groups.
process_output_of	List	Yes	Defines the source of the alerts that Tempus processes. You can specify none, one or more Moolets. Typically Tempus processes the output of its direct upstream neighbor in the processing chain. Usually this is "Alert Workflows" which are the output from the Alert Workflow Engine.
run_on_startup	Boolean	No	Whether this Tempus algorithm should start when Moogfarmd starts.
partition_by	String	No	Splits clustering according to the entered component. After alerts have been clustered and before they enter merging and resolution, you can split clusters into sub-clusters based on a component of the events. For example, you can use the **manager** parameter to ensure that Situations only contain events from the same manager. Note Cisco does not recommend partitioning by components.
pre_partition	Boolean	No	Partitions event streams before clustering. You specify a component field on which the event stream will be partitioned before clustering occurs. The alerts in the resulting Situations each contain a single value for the component field

			chosen.
significance_test	String	No	Calculation that determines how significant a cluster of alerts or a potential Situation must be for Tempus to detect it. **Poisson1**, looks at the data of a single alert cluster to calculate how significant it is. This more likely to detect all significant alert clusters but with a higher risk of creating insignificant alert clusters. Use this option when your alerts originate from different networks or unrelated topologies. **Poisson2** is a more thorough test that looks at an alert cluster and all alerts outside the cluster with a similar event rate. It is more likely to exclude all insignificant alert clusters but with a high risk of excluding significant alert clusters. Use this option if you expect all of your alerts to come from the same connected network. See Poisson distribution for more information.
significance_threshold	Number	No	Sets the maximum significance score for Tempus to create a Situation. The score is proportional to the probability that the alert cluster or potential Situation was coincidence. The lower the score, the more significant the cluster and the least likely it was a coincidence. This score ranges from 0 to 100.
detection_algorithm	String	No	Detection algorithm that Tempus uses, one of: **Louvain**, **LouvainMulti**, or **SmartLocal**.

Response

Endpoint **updateTempus** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

This endpoint returns an error code if the values of **entropy_threshold** and **threshold_type** are inconsistent. For example, if the **entropy_threshold** is set to 0.4 and **threshold_type** is set to global.

Examples

The following examples demonstrate typical use of endpoint **updateTempus**:

Request examples

Example cURL request to update the detection algorithm on Tempus algorithm 'newTempus':

```
curl -X POST -u graze:graze -k "https://localhost/graze/v1/updateTempus" -H
"Content-Type: application/json; charset=UTF-8" --data \
'{ \
"name": "newTempus", \
"detection_algorithm": "LouvainMulti" \
}'
```

Example cURL request to update Tempus algorithm 'newTempus' to use the global entropy threshold in Cisco Crosswork Situation Manager:

```
curl -X POST -u graze:graze -k "https://localhost/graze/v1/updateTempus" -H
"Content-Type: application/json; charset=UTF-8" --data \
'{ \
"name": "newTempus", \
"threshold_type": "global" \
}'
```

Example cURL request to update Tempus algorithm 'newTempus' to use an explicit entropy threshold of 0.22:

```
curl -X POST -u graze:graze -k "https://localhost/graze/v1/updateTempus" -H
"Content-Type: application/json; charset=UTF-8" --data \
'{ \
"name": "newTempus", \
"entropy_threshold": 0.22, \
"threshold_type": "explicit_value" \
}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateUser

A Graze API POST request that updates an existing user.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`updateUser`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`username`	String	No, you use if **`uid`**.	Username of the user to be updated.
`uid`	Long	No, if you use **`username`**.	User ID of the user to be updated.
`password`	String	No	New user password, only valid for DB realm.
`active`	Boolean	No	Set to **`true`** if the user is active, **`false`** if the user is inactive. Default is **`true`**.
`email`	String	No	User's email address.
`fullname`	String	No	User's full name.
`roles`	JSON list	No	List of either the role IDs or the role names. For

			example, **"roles":["Super User"]**.
primary_group	String or Number	No	User's primary group name or primary group ID.
department	String or Number	No	User's department ID or department name.
timezone	String	No	User's timezone.
contact_num	String	No	User's phone number.
session_expiry	Number	No	Number of minutes after which the user's session expires. Default is the system default.
competencies	JSON list	No	A list with the user competencies. Each competency should have have name or cid and ranking. For example: **[** **{"name":"SunOS", "ranking": 40},** **{"name":"SAP", "ranking": 50},** **{"name":"EMC", "ranking": 60}** **]**
teams	JSON list of Numbers or Strings	No	List of the user's team names or team IDs.

Response

Endpoint **updateUser** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateUser**:

Request example

Example cURL request to update user ID 5:

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateUser" \
-H "Content-Type: application/json; charset=UTF-8" \
-d '{"uid" : 5, "active" : true, "password" : "test", "roles" : ["Super User",
"Operator"], "teams" : ["my team 1"], "session_expiry" : null, "properties" :
null, "contact_num" : "555-123456", "timezone" : "Europe/London", "fullname" :
"John Doe", "department" : "Support", "primary_group" : "Network", "email" :
"test@test.com"}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateValueRecipe

A Graze API POST request that updates a Cookbook Recipe that uses either a Value Recipe or a Value Recipe v2 recipe type. See Recipe Types.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **updateValueRecipe** takes the following request arguments. You must supply the name of the Cookbook Recipe plus at least one other argument that you want to change.

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
name	String	Yes	Name of the Recipe that you want to update.
cookbook	List of Strings	No	A list of the Cookbooks that this Recipe belongs to.
description	String	No	Description of the Recipe.
version	String	No	Defines whether the Recipe uses Value Recipe or Value Recipe v2. Valid values are **V1** for the Value Recipe and **V2** for Value Recipe v2. Default is **V2**. See Recipe Types for more information. Use updateBotRecipe if you want to update a Bot Recipe.
alert_threshold	Positive Integer	No	Minimum number of alerts required before Cookbook creates a Situation. When Cookbook determines the number of alerts required to create a Situation, it compares the alert threshold values in the Cookbook Recipe and in the merge group that the Cookbook Recipe belongs to, and it uses the higher value. If you are using the default merge group which has an alert threshold of 2, Cookbook will never create a Situation containing a single alert. If you want Cisco Crosswork Situation Manager to create Situations with a single alert, consider changing the alert threshold in the default merge group to 1 or creating custom merge groups. See Merge Groupsfor more information on updating the default merge group and setting up custom merge groups.
trigger	String	No	A filter that determines the alerts that Cookbook considers for Situation creation. Cookbook includes alerts that match the trigger filter. By default Cookbook only includes alerts with a severity of 'Critical'.
exclusion	String	No	A filter that determines the alerts to exclude from Situation creation. Cookbook ignores alerts that match the exclusion filter.

seed_alert	String	No	A filter that determines whether to create a Situation from a seed alert. The seed alert must meet both the **trigger**, **exclusion** and **seed_alert** criteria to create a Situation. Cookbook considers subsequent alerts for clustering if they meet the trigger and exclusion filter criteria. Alerts that arrive prior to the seed alert that met the trigger and exclusion filter criteria do not form Situations.
rate	Positive Integer	No	Rate, in number of alerts per second. Cookbook clusters alerts if they arrive at a higher rate than is specified here. Cookbook uses **rate** together with **min_sample_size** and **max_sample_size** to determines whether to cluster alerts into Situations. See Cookbook and Recipe Examples.
min_sample_size	Positive Integer	No	Number of alerts that must arrive before the Cookbook starts to calculate the alert rate. See Cookbook and Recipe Examples.
max_sample_size	Positive Integer	No	Maximum number of alerts that are considered in the alert rate calculation. When more than this number of alerts have arrived, Cookbook discards the oldest alerts and calculates the alert rate based on the number of alerts in the **max_sample_size**. See Cookbook and Recipe Examples. Valid only if **rate** is non-zero.
cook_for	Positive Integer	No	Minimum time period, in seconds, that the Cookbook Recipe clusters alerts for before it resets and starts a new cluster. See Cookbook and Recipe Examples. If you set a different **cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for** time inherit the value from the Cookbook. Inherits value from Cookbook if omitted.
cook_for_extension	Positive Integer	No	Time period that the Cookbook Recipe can extend clustering alerts for before it resets and starts a new cluster. Setting this value enables the cook for auto-extension feature for this Cookbook. As Cookbook receives related alerts, it continues to extend the total clustering time until the **max_cook_for** period is reached. Used in conjunction with the **max_cook_for** value, the **cook_for_extension** period helps to ensure that Cookbook continues to cluster alerts together that are related to the same failure. The **cook_for_extension** period only applies to new related alerts; it does not apply to existing alerts that are updated with new events. See Cookbook and Recipe Examples.

			If you set a different **cook_for_extension** time for a Recipe, it overrides the Cookbook value. Recipes without a **cook_for_extension** time inherit the value from the Cookbook. Inherits value from Cookbook if omitted.
max_cook_for	Positive Integer	No	Maximum time period that the Cookbook Recipe clusters alerts for before it resets and starts a new cluster. It works in conjunction with the **cook_for_extension** time to help ensure that Cookbook continues to cluster alerts together that are related to the same failure. This value is ignored unless the **cook_for_extension** time is specified. See Cookbook and Recipe Examples. If you set a different **max_cook_for** time for a Recipe, it overrides the Cookbook value. Recipes without a **max_cook_for** value inherit the value from the Cookbook. Inherits value from Cookbook if omitted.
cluster_by	String	No	Determines Cookbook's clustering behavior. Set to an empty string to use the Cookbook **cluster_by** setting. Set to **first_match** so that Cookbook adds alerts to the first cluster over the similarity threshold value. Set to **closest_match** to add alerts to the cluster with the highest similarity greater than the similarity threshold value. This option may be less efficient because Cookbook needs to compare alerts against each cluster in a Recipe. Set to an empty string to use the Cookbook setting. If you set a different **cluster_by** value for a Recipe, it overrides the Cookbook value. Recipes without a **cluster_by** value inherit the value from the Cookbook.
hop_limit	Positive Integer	No	Maximum number of hops between the alert source nodes in order for the alerts to quality for clustering. Cisco Crosswork Situation Manager measures hop limit from the first alert that formed the Situation and always follows the shortest possible route. A hop is the distance between two directly connected nodes. You can only set a hop limit if you have one or more topologies in your system. For more information on hops and hop limit see Vertex Entropy and Configure Topology-based Clustering with Vertex Entropy. For more information on topologies see

			Topologies.
components	JSON Array	No	Values that alerts must match for Cookbook to include them in a Situation. You can provide values for multiple components. See the table below for a full description of all components.
use_dynamic_topology	Boolean	No	Infer the topology to cluster on from the **moog_topology** field in the alert's custom info. If you use a dynamic topology you cannot set **topology_name**.
alert_matching_attribute	String	No	The alert field that specifies the topology node from which the alert was generated. If you set an alert matching attribute you must set **dynamic_topology** to **true** or set the **topology_name**.
topology_name	String	No	Restrict clustering to nodes in the specified topology. If you set a topology name you cannot set **dynamic_topology** to **true**.

The **components** property is an array of JSON objects containing the following:

Name	Type	Required	Description
name	String	Yes	Name of the component.
similarity	Double	Yes	Similarity threshold that the component must meet for Cookbook to cluster the alert into a Situation.
shingle_size	Integer	No	Shingle size for Cookbook to use to determine the similarity between different strings. The shingle size is only valid for Recipe Value v2 recipes. Default is -1 which means that Cookbook uses words to determine similarity. See Recipe Types.
treat_as	String	No	Determines whether Cookbook treats the component as a string or matches each value in the list individually. See Recipe Types for details. Valid values are **List** and **String**. Default is **String**.
case_sensitive	Boolean	No	Enables or disables case sensitive when comparing strings. Case sensitivity is only valid for Recipe Value recipes. See Recipe Types for more details. Default is **true** which means that strings are treated as case sensitive.

Response

Endpoint **updateValueRecipe** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **updateValueRecipe**:

Request example

Example cURL request to update Value Recipe "GrazeRecipe":

```
curl -X POST -u graze:graze -k -v "https://localhost/graze/v1/updateValueRecipe"
-H "Content-Type: application/json; charset=UTF-8" --data '{
    "name" : "GrazeRecipe",
    "dynamic_topology" : false,
    "topology_name" : "physical",
    "hop_limit" : 2,
    "components" : [{"name": "component_name", "similarity":0.8, "shingle_size"
: 3}]
}'
```

Response example

A successful request returns the HTTP code 200 and no response text.

updateWorkflow

A Graze API POST request that updates an existing workflow in the Workflow Engine.

Back to Graze API EndPoint Reference.

Request arguments

Endpoint **`updateWorkflow`** takes the following request arguments:

Name	Type	Required	Description
`auth_token`	String	Yes	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`id`	String	Yes	ID of the workflow that you want to update.
`workflow_name`	String	No	Name of the workflow.
`active`	Boolean	No	Determines whether the workflow is active or not. If **`true`**, the workflow is active.
`description`	String	No	Description of the workflow.
`entry_filter`	String	No	An SQL-like filter to determine which events, alerts or Situations can enter the workflow. If empty, the workflow accepts all events, alerts or Situations.
`sweep_up_filter`	String	No	An SQL-like filter to intake any additional events, alerts or Situations from the database.
`first_match_only`	Boolean	No	If enabled, events, alerts, and Situations only pass through actions on the first time they enter the Workflow Engine.
`operations`	JSON Array	No	List of properties relating to each operation: Name — Type — Required — Description type — String — Yes — Type of operation. Options are: 'action', 'decision' and

						'delay'.
			operation_name	String	Yes, for 'action' and 'decision' types.	Name of the operation.
			function_name	String	Yes, for 'action' and 'decision' types.	Name of the function.
			forwarding_behavior	String	No	Forwarding behavior for the function. One of:**`always forward`**: The function always forwards the object to the next workflow.**`stop this workflow`**: The function stops this workflow and the object moves to the next workflow.**`stop all workflows`**: The function stops all workflows for this object.Default is **`always forward`**. Only valid for 'action' and 'decision' types.
			function_args	JSON Object	No	Arguments for the function.
			duration	Integer	Yes, for 'delay' type.	Length of time before the message goes to the next

			reset	Boolean	Yes, for 'delay' type.	operation. Determines whether the timer resets after each occurrence.

Response

Endpoint **`updateWorkflow`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **`updateWorkflow`**:

Request examples

Example cURL request to deactivate workflow ID 14:

```
curl -X POST -u graze:graze -k \
-v "https://localhost/graze/v1/updateWorkflow" \
-H "Content-Type: application/json; charset=UTF-8" \
--data '{ "id" : 14,"active" :false }'
```

Example cURL request to rename workflow ID 14:

```
curl -X POST -u graze:graze -k \
-v "https://localhost/graze/v1/updateWorkflow" \
-H "Content-Type: application/json; charset=UTF-8" \
--data '{ "id" : 14, "workflow_name" : "Deactivated Example" }'
```

Example cURL request to update the entry filter and sweep-up filter for workflow ID 3:

```
curl -X POST -u graze:graze -k \
-v "<https://192.168.56.10/graze/v1/updateWorkflow"> \
-H "Content-Type: application/json; charset=UTF-8" \
--data '{
    "id" : 3,
    "entry_filter": "state = 8",
    "sweep_up_filter": "state = 8"
    }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

Alert Action Codes

The Graze API endpoint and MoogDb V2 method **`getAlertActions`** can retrieve actions that happened on a given alert.

The table below shows the list of IDs and the matching description for each action:

Event ID	Description
0	Alert Created
2	Event Added to Alert
3	Alert Assigned
4	Alert Updated
5	Alert Updated Custom Info
6	Alert Added to Situation
7	Team Updated
8	Alert Resolved
9	Alert Closed
10	Ran Tool

Situation Action Codes

The getSituationActions Graze API endpoint and MoogDb V2 method can retrieve actions that happened on a given Situation.

The table below shows the list of IDs and the matching description for each action:

Event Id	Description
1	Situation Created
2	Assigned Moderator
3	Situation Resolved
4	Situation Revived
5	Situation Closed
6	Assigned Queue
7	Created By Merge
8	Used In Merge
9	Created By Split
10	Used For Split
11	Ran Tool
12	Acknowledged Situation Moderator
13	Deacknowledged Situation Moderator
14	Added Alerts To Situation
15	Added Entry To Thread

16	Changed Situation Processes
17	Changed Situation Services
18	Created Thread
19	Agreed With Thread Entry
21	Commented On Thread Entry
22	Disagreed With Thread Entry
23	Changed Situation Custom Info
24	Described Situation
25	Excluded User
26	Invited User
27	Moved Alerts To Situation
28	Removed Alerts From Situation
29	Situation Updated
30	Situation Teams Changes
31	Marked Thread Entry As Resolving
32	Unmarked Thread Entry As Resolving
33	Situation Rated
34	Situation Rating Removed
35	Situation Internal Severity Changed
36	Situation Superseding Others
37	Updated Comment On Thread Entry
38	Updated Entry Of Thread

Situation Flags

You can use Situation flags to determine actions that users have performed on Situations, such as adding a manual description to a Situation or manually assigning a Situation to a team.

The table below shows the list of codes and the matching description for each Situation flag available in Cisco Crosswork Situation Manager:

Flag Code	Description
1	LEAVE_MANUAL_DESCRIPTION

2	MANUALLY_ASSIGNED_TO_TEAM

You can use the following Graze API endpoints and MoogDb V2 methods to create more Situation flags and associate them with Situations. For example, you may want to set up a flag for "TICKETED" when a ticket has been raised for a Situation.

- **checkSituationFlag**: Checks whether a flag is associated with a Situation, in Graze API and MoogDb V2.
- **getSituationFlags**: Returns the flags for one or an array of Situations, in Graze API and MoogDb V2.
- **getSituationsWithFlag**: Returns all the Situations which have the specified flag, in Graze API and MoogDb V2.
- **setSituationFlags**: Updates the flags associated with a specified Situation, in Graze API and MoogDb V2.

API Update Behavior

The behavior of the Graze API endpoints and MoogDb V2 methods depends on the status of the Situation they are acting on. The three relevant statuses are:

- Open Situation: The Situation is open.
- Closed Situation in active database: For a period of time after the Situation has been closed, it remains in the active database. This period of time is known as the "grace period".
- Closed Situation in historic database: After the grace period has expired, the Situation is moved to the historic database.

Each API endpoint/method topic describes its behavior in these three Situation statuses.

See Configure Historic Data Retention for more information on the active and historic databases.

Stats API

You can use the Stats API endpoints to report on Cisco Crosswork Situation Manager data. These endpoints return various statistics about teams, Situations and services.

You can also fetch information on the Mean Time to Acknowledge (MTTA), Mean Time to Detect (MTTD) and Mean Time to Resolve (MTTR).

System Endpoints

The following endpoints return data statistics relating to your Cisco Crosswork Situation Manager system:

- getAlertsInNewSituationsStats: Returns the number of alerts that belong to new Situations in the specified time range.
- getMTTAStats: Returns the Mean Time To Acknowledge (MTTA) Situations in the specified time range.
- getMTTDStats: Returns the Mean Time To Detect (MTTD) Situations in the specified time range.
- getMTTRStats: Returns the Mean Time To Resolve (MTTR) for Situations in the specified time range.
- getNewAlertsStats: Returns the number of new alerts in the specified time range.

- getNewAlertsPerSituationsStats: Returns the percentage of noise reduction from alerts-to-Situations clustering in the specified time range.
- getNewEventsPerAlertsStats: Returns the percentage of noise reduction from events-to-alerts aggregation and deduplication in the specified time range.
- getNewEventsPerSituationsStats: Returns the percentage of noise reduction from events-to-Situations aggregation, deduplication, and clustering in the specified time range.
- getNewSituationsStats: Returns the number of new Situations created in the specified time range.
- getReassignedSituationStats: Returns the number of Situations reassigned in the specified time range.
- getReoccurringSituationStats: Returns the percentage of reoccurring situations in the specified time range.
- getServiceSituationStats: Returns the number of active Situations impacting a service in the specified time range.
- getSeveritySituationStats: Returns the number of Situations by severity in the specified time range.
- getStats: Returns all available Stats API endpoints along with their description and request parameters.
- getStatusSituationStats: Returns the number of Situations by status.
- getSystemSituationStats: Returns the number of active Situations in the specified time range.
- getTopServiceSituationStats: Returns the number of active Situations impacting a top service in the specified time range.

Team Endpoints

The following endpoints return data statistics relating to your Cisco Crosswork Situation Manager teams:

- getCommentCountPerTeamStats: Returns the total number of comments each hour for a specific team or teams in the specified time range.
- getMTTAPerTeamStats: Returns the mean time to acknowledge (MTTA) a Situation per team in the specified time range.
- getMTTRPerTeamStats: Returns the mean time to resolve (MTTR) a Situation per team in the specified time range.
- getReassignedSituationsPerTeamStats: Returns the number of reassigned Situations associated with a team or multiple teams in the specified time range.
- getReoccurringSituationPerTeamStats: Returns the number of reoccurring Situations associated with a team in the specified time range.
- getServiceSituationPerTeamStats: Returns the number of Situations impacting each service for a team.
- getSeveritySituationPerTeamStats: Returns the number of Situations by severity per team in the specified time range.

- getStatusSituationPerTeamStats: Returns the number of Situations by status for a team in the specified time range.
- getTeamSituationStats: Returns the number of active Situations assigned to a team in the specified time range.
- getTopServiceSituationStats: Returns the number of active Situations impacting a top service in the specified time range.

User Endpoints

The following endpoints return data statistics relating to your Cisco Crosswork Situation Manager users:

1. getAlertsMarkedPRCPerUserStats: Returns the total number of alerts marked with probable root cause (PRC) feedback by each user.
2. getAcknowledgedSituationsPerUserStats: Returns the number of Situations acknowledged by a specific user or users in the specified time range.
3. getAssignedSituationsPerUserStats: Returns the number of Situations assigned to a specific user or users in the specified time range.
4. getChatOpsToolExecutedPerUserStats: Returns the number of ChatOps tools executed by a user each hour in the specified time range.
5. getClosedSituationsPerUserStats: Returns the number of Situations that a user has closed each hour in the specified time range.
6. getCommentCountPerUserStats: Returns the number of comments left by a user or users in the specified time range.
7. getInvitationsReceivedPerUserStats: Returns the number of Situation invitations received for a given user each hour in the specified time range.
8. getMTTAPerUserStats: Returns the mean time it takes a user to acknowledge a Situation in the specified time range.
9. getMTTRPerUserStats: Returns the mean time it takes a user to resolve a Situation in the specified time range.
10. getOpenSituationsPerUserStats: Returns the number of open Situations assigned to a user at each data point.
11. getRatedSituationsPerUserStats: Returns the number of Situations rated by a user in the specified time range.
12. getReassignedSituationsPerUserStats: Returns the number of Situations reassigned by a user in the specified time range.
13. getResolvedSituationsPerUserStats: Returns the number of Situations resolved by a user in the specified time range.
14. getViewedSituationsPerUserStats: Returns the number of Situations a user has viewed in the specified time range.
15. getWorkedSituationsPerUserStats: Returns the number of Situations a user has worked on in the specified time range.

getAcknowledgedSituationsPerUserStats

A GET request that returns the number of Situations acknowledged by a specific user or users within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getAcknowledgedSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getAcknowledgedSituationsPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Acknowledged Situations (full name)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations acknowledged by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations acknowledged each hour in the time period. 1 week to 1 month: Returns the number of Situations acknowledged each day in the time period. 1 month to 1 year: Returns the number of Situations acknowledged each

		week in the time period. More than 1 year: Returns the number of Situations acknowledged each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getAcknowledgedSituationsPerUserStats`**:

Request example

A cURL request to return the number of Situations acknowledged by user Bob from 9am on Friday 28th September until 3pm on Friday 28th September 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getAcknowledgedSituationsPerUserStats" --data-
urlencode 'users=[6]' --data-urlencode 'from=1538121620' --data-urlencode
'to=1538143220'
```

Response example

A successful response returns the number of Situations acknowledged by Bob each hour during that time frame:

```
[{
        "datapoints":[
                [2.0,1538121620000],
                [3.0,1538125220000],
                [0.0,1538128820000],
                [2.0,1538132420000],
                [2.0,1538136020000],
                [2.0,1538139620000]
        ],
        "target":"Acknowledged Situations (Bob Bowden)"
}]
```

getAlertsInNewSituationsStats

A GET request that returns the number of alerts that belong to new Situations during the specified time range.

Back to Stats API.

Request arguments

Endpoint **`getAlertsInNewSituationsStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time.

		none: No aggregation of data points.

Response

Endpoint **`getAlertsInNewSituationsStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Alerts in new situations"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of alerts. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of alerts each hour in the time period.1 week to 1 month: Returns the number of alerts each day in the time period.1 month to 1 year: Returns the number of alerts each week in the time period.More than 1 year: Returns the number of alerts each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getAlertsInNewSituationsStats`**:

Request example

A cURL GET request for all alerts in new Situations over a 24 hour time range from 13.23pm on Tuesday 18th September until 13:24pm on Wednesday 19th September 2018::

```
curl -G -u graze:graze -k -v
"https://freida7/graze/v1/getAlertsInNewSituationsStats" --data-urlencode
'from=1537277017' --data-urlencode 'to=1537363453'
```

Response example

A successful response indicating there were 56 alerts at13:23pm on Wednesday 19th September 2018:

```
[
        {"datapoints":[
        [56.0,1537359817000]
        ],
        "target":"Alerts in new situations"}
]
```

getAlertsMarkedPRCPerUserStats

A GET request that returns the total number of alerts marked with probable root cause (PRC) feedback by each user.

Back to Stats API.

Request arguments

Endpoint **`getAlertsMarkedPRCPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getAlertsMarkedPRCPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Alerts Marked PRC (full name)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of alerts marked with PRC feedback by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of alerts marked with PRC feedback each hour in the time period. 1 week to 1 month: Returns the number of alerts marked with PRC feedback each day in the time period. 1 month to 1 year: Returns the number of alerts marked with PRC feedback each week in the time period. More than 1 year: Returns the number of alerts marked with PRC feedback each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getAlertsMarkedPRCPerUserStats`**:

Request example

A cURL request to return the number of alerts marked with PRC feedback to users 5 and 6 from 8am until 2pm on on Friday, 28th September 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getAlertsMarkedPRCPerUserStats" --data-urlencode
'users=[5, 6]' --data-urlencode 'from=1538121620' --data-urlencode
'to=1538143220' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of alerts that users Alice and Ian have marked with PRC feedback each hour during the time range:

```
[{
        "datapoints":[
                [22.0,1538121620000],
                [18.0,1538125220000],
                [30.0,1538128820000],
                [23.0,1538132420000],
                [29.0,1538136020000],
                [28.0,1538139620000]]
        ],
        "target":"Alerts Marked PRC (Alice Anderson)"
}
{
        "datapoints":[
                [34.0,1538121620000],
                [20.0,1538125220000],
                [35.0,1538128820000],
                [21.0,1538132420000],
                [19.0,1538136020000],
                [10.0,1538139620000]]
        ],
        "target":"Alerts Marked PRC (Ian Ince)"
}]
```

getAssignedSituationsPerUserStats

A GET request that returns the number of Situations assigned to a specific user or users within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getAssignedSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.

from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
aggregation	String	Set to: **accumulate**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getAssignedSituationsPerUserStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Assigned Situations (full name)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations assigned to the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations assigned each hour in the time period.1 week to 1 month: Returns the number of Situations assigned each day in the time period.1 month to 1 year: Returns the number of Situations assigned each week in the time period.More than 1 year: Returns the number of Situations assigned each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getAssignedSituationsPerUserStats**:

Request example

A cURL request to return the number of Situations assigned to users 10 and 11 from 11pm on Tuesday, 25th September 2018 until 11pm on Wednesday, 26th September 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getAssignedSituationsPerUserStats" --data-urlencode
'users=[10,11]' --data-urlencode 'from=1537916400' --data-urlencode
'to=1538002799' --data-urlencode 'aggregation=sum'
```

Response example

A successful response returns the number of Situations assigned to the users Frank and Dave each hour during the time range:

```
[{
        "datapoints":[
                [10.0,1537916400000],
```

```
                [5.0,1537920000000],
                [7.0,1537923600000],
                [7.0,1537927200000],
                [7.0,1537930800000],
                [1.0,1537934400000],
                [5.0,1537938000000],
                [6.0,1537941600000],
                [9.0,1537945200000],
                [9.0,1537948800000],
                [7.0,1537952400000],
                [8.0,1537956000000]
            ],
            "target":"Assigned Situations (Frank Fuller/Dave Danton)"
    }]
```

getChatOpsToolExecutedPerUserStats

A GET request that returns the number of ChatOps tools executed by a user each hour within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getChatOpsToolExecutedPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getChatOpsToolExecutedPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description

target	String	"Chat Ops Tools executed (full name)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of ChatOps tools executed by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of ChatOps tools executed each hour in the time period. 1 week to 1 month: Returns the number of ChatOps tools executed each day in the time period. 1 month to 1 year: Returns the number of ChatOps tools executed each week in the time period. More than 1 year: Returns the number of ChatOps tools executed each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getChatOpsToolExecutedPerUserStats**:

Request example

A cURL request to retrieve the total number of ChatOps tools executed by user 5 from 11pm on Sunday, 14th October until 11pm on Monday, 15th October 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getChatOpsToolExecutedPerUserStats" --data-urlencode
'users=[5]' --data-urlencode 'from=1539558000' --data-urlencode 'to=1539644399'
--data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of ChatOps tools executed by the user Max each hour:

```
[{
        "datapoints":[
                [6.0,1539558000000],
                [24.0,1539561600000],
                [1.0,1539565200000],
                [0.0,1539568800000],
                [14.0,1539572400000],
                [10.0,1539576000000],
                [4.0,1539579600000],
                [12.0,1539583200000],
                [25.0,1539586800000],
                [8.0,1539590400000],
                [0.0,1539598043846]
        ],
                "target":"ChatOps Tools executed (Max Matthews)"
}]
```

getClosedSituationsPerUserStats

A GET request that returns the number of Situations that a user has closed each hour within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getClosedSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getClosedSituationsPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Closed Situations (full name)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations closed by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number ofSituations closed each hour in the time period. 1 week to 1 month: Returns the number of Situations closed each day in the time period. 1 month to 1 year: Returns the number of Situations closed each week in the time period. More than 1 year: Returns the number of Situations closed each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getClosedSituationsPerUserStats`**:

Request example

A cURL request to return the number of Situations closed by user 5 from 6am until midnight on October 1st 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getClosedSituationsPerUserStats" --data-urlencode
'users=[5]' --data-urlencode 'from=1538373600' --data-urlencode 'to=1538395200'
--data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of Situations closed by user Chris each hour during the time range:

```
[{
        "datapoints":[
                [1.0,1539558000000],
                [1.0,1539561600000],
                [2.0,1539565200000],
                [5.0,1539568800000],
                [0.0,1539572400000],
                [7.0,1539576000000],
                [1.0,1539579600000],
                [0.0,1539583200000],
                [8.0,1539586800000],
                [6.0,1539590400000],
                [0.0,1539594000000],
                [0.0,1539597600000]
        ],
        "target":"Closed Situations (Chris Collins)"
}]
```

getCommentCountPerTeamStats

A GET request that returns the total number of comments each hour for a specific team or teams in a given time range.

Back to Stats API.

Request arguments

Endpoint **`getCommentCountPerTeamStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`teams`	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

aggregation	String	Set to: **accumulate**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getCommentCountPerTeamStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	Name of the team.
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of comments. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of comments each hour in the time period.1 week to 1 month: Returns the number of comments each day in the time period.1 month to 1 year: Returns the number of comments each week in the time period.More than 1 year: Returns the number of comments each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getCommentCountPerTeamStats**:

Request example

A cURL request to retrieve the total number of comments for three teams each hour over a 24 hour time range from 6am on Wednesday 19th September until 6am on Thursday 20th September 2018:

```
curl -G -u graze:graze -k -v
"https://freida7/graze/v1/getCommentCountPerTeamStats" --data-urlencode
'teams=[1,2,3]' --data-urlencode 'from=1537336800' --data-urlencode
'to=1537423200' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of comments per hour for the Cloud DevOps, Database DevOps and Switch DevOps teams:

```
[
        {"datapoints":[
                [14.0,1537357717000]],
                "target":"Cloud DevOps"},
        {"datapoints":[
                [22.0,1537357717000]],
```

```
                "target":"Database DevOps"},
        {"datapoints":[
                [10.0,1537357717000]],
                "target":"Switch DevOps"}
]
```

getCommentCountPerUserStats

A GET request that returns the number of comments left by a user or users within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getCommentCountPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getCommentCountPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Number of Comments (full name)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of comments. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of comments each hour in the time period. 1 week to 1 month: Returns the number of comments each day in the time period.

		1 month to 1 year: Returns the number of comments each week in the time period. More than 1 year: Returns the number of comments each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getCommentCountPerUserStats`**:

Request example

A cURL request to retrieve the total number of comments made by users 9 and 11 each hour from 11pm on Sunday, 14th October until 11pm on Monday, 15th October 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getCommentCountPerUserStats" --data-urlencode
'users=[9,11]' --data-urlencode 'from=1539558000' --data-urlencode
'to=1539644399' --data-urlencode 'aggregation=sum'
```

Response example

A successful response returns the number of comments made by the users Ian and Sharon each hour:

```
[{
        "datapoints":[
                [6.0,1539558000000],
                [24.0,1539561600000],
                [1.0,1539565200000],
                [0.0,1539568800000],
                [14.0,1539572400000],
                [10.0,1539576000000],
                [4.0,1539579600000],
                [12.0,1539583200000],
                [25.0,1539586800000],
                [8.0,1539590400000],
                [0.0,1539598043846]
        ],
                "target":"Number of Comments (Ian Ince/Sharon Scott)"
}]
```

getInvitationsReceivedPerUserStats

A GET request that returns the number of Situation invitations received for a given user each hour within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getInvitationsReceivedPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any

		data.
from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
aggregation	String	Set to: **accumulate**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getInvitationsReceivedPerUserStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Invitations Received (full name)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Invitations received by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situation invitations each hour in the time period. 1 week to 1 month: Returns the number of Situation invitations each day in the time period. 1 month to 1 year: Returns the number of Situation invitations each week in the time period. More than 1 year: Returns the number of Situation invitations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getInvitationsReceivedPerUserStats**:

Request example

A cURL request for the number of Situation invitations for users 7 and 8 from midnight on Sunday, 14th October until 6am on Monday, 15th October 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getInvitationsReceivedPerUserStats" --data-urlencode
'users=[7,8]' --data-urlencode 'from=1539558000' --data-urlencode
'to=1539583200' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of invitations for users 7 and 8:

```
[{
        "datapoints": [
                [1.0,1539558000000],
                [1.0,1539561600000],
                [2.0,1539565200000],
                [5.0,1539568800000],
                [0.0,1539572400000],
                [7.0,1539576000000],
                [1.0,1539579600000],
                [0.0,1539583200000],
                [8.0,1539586800000],
                [1.0,1539579600000],
                [2.0,1539583200000],
                [0.0,1539586800000],
        ],
        "target": "Invitations Received (Peter Parker/Kat Knight)"
}]
```

getMTTAPerTeamStats

A GET request that returns the mean time to acknowledge (MTTA) a Situation per team in the specified time range.

Back to Stats API.

Request arguments

Endpoint **`getMTTAPerTeamStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`teams`	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **`getMTTAPerTeamStats`** returns the following response:

Type	Description

HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Mean Time to Acknowledge (MTTA)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point:MTTA (seconds) for that bucket. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the MTTA each hour in the time period.1 week to 1 month: Returns the MTTA each day in the time period.1 month to 1 year: Returns the MTTA each week in the time period.More than 1 year: Returns the MTTA each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getMTTAPerTeamStats**:

Request example

A cURL command request to find out the MTTA for the Cloud DevOps team over a year from 13.14pm on Monday 31st July 2017 until 13.14.pm on Tuesday 31st July 2018:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getMTTAPerTeamStats" --
data-urlencode 'from=1501506840' --data-urlencode 'to=1533042840' --data-
urlencode 'teams=[1]'  --data-urlencode 'aggregation=none'
```

Response example

A successful response shows the MTTA for the year was 3.32 minutes:

```
[{
 "datapoints": [
        [213.0, 1532956486000]
 ],
 "target": "Mean Time to Acknowledge (MTTA)"
}]
```

getMTTAPerUserStats

A GET request that returns the mean time it takes a user to acknowledge a Situation within a given time range.

Back to Stats API.

Request arguments

Endpoint **getMTTAPerUserStats** takes the following request arguments.

Name	Type	Description
auth_token	String	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

users	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
aggregation	String	Set to: **accumulate**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getMTTAPerUserStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Mean Time to Acknowledge (MTTA) (full name)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point:MTTA (seconds) for that bucket. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of MTTA each hour in the time period. 1 week to 1 month: Returns the number of MTTA each day in the time period. 1 month to 1 year: Returns the number of MTTA each week in the time period. More than 1 year: Returns the number of MTTA each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getMTTAPerUserStats**:

Request example

A cURL request for the MTTA for user 5 from 6.34am until 2.35pm on Tuesday, 25th September 2018:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getMTTAPerUserStats" --
data-urlencode 'users=[5]' --data-urlencode 'from=1537857295' --data-urlencode
'to=1537886111' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the MTTA each hour for the user Robert:

```
[{
        "datapoints": [
                [221,1537857295000],
                [960,1537860895000],
                [901,1537864495000],
                [1196,1537868095000],
                [671,1537871695000],
                [1241,1537875295000],
                [556,1537878895000]
        ],
        "target": "Mean Time to Acknowledge (MTTA)(Robert Richards)"
}]
```

getMTTAStats

A GET request that returns the Mean Time To Acknowledge (MTTA) Situations in the specified time range.

The time to acknowledge (TTA) for a Situation is the duration from the first event's inclusion in the Situation to the time when a moderator assigns a Situation to a user in Moogsoft AIOps.

Back to Stats API.

Request arguments

Endpoint **`getMTTAStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

Response

Endpoint **`getMTTAStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Mean Time to Acknowledge (MTTA)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: MTTA (seconds) for that bucket. Timestamp: Calculation time (Unix epoch timestamp in milliseconds).

		Less than 1 week: Returns the MTTA each hour in the time period.1 week to 1 month: Returns the MTTA each day in the time period.1 month to 1 year: Returns the MTTA each week in the time period.More than 1 year: Returns the MTTA each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getMTTAStats`**:

Request example

A cURL command to return the MTTA for Moogsoft AIOps over a 24 hour time range from 11.09am on Sunday 17th December until 11.09am on Monday 18th December 2017:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getMTTAStats" --data-
urlencode 'from=1513508950' --data-urlencode 'to=1513595370'
```

Response example

A successful response returns the MTTA in seconds for each hour:

```
[{
        "datapoints": [
                [312.0, 1513657700000],
                [209.0, 1513661300000],
                [101.0, 1513664900000],
                [114.0, 1513668500000],
                [203.0, 1513672100000],
                [120.0, 1513675700000],
                [201.0, 1513679300000],
                [90.0, 1513682900000],
                [100.0, 1513686500000]
        ],
        "target": "Mean Time to Acknowledge (MTTA)"
}]
```

getMTTDStats

A GET request that returns the Mean Time To Detect (MTTD) Situations in the specified time range.

The time to detect (TTD) for a Situation is the duration from the first event's inclusion in the Situation to the Situation creation time.

Back to Stats API.

Request arguments

Endpoint **`getMTTDStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint

		returned is the current state datapoint.

Response

Endpoint **`getMTTDStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Mean Time to Detect (MTTD)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: MTTD (seconds) for that bucket Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the MTTD each hour in the time period.1 week to 1 month: Returns the MTTD each day in the time period.1 month to 1 year: Returns the MTTD each week in the time period.More than 1 year: Returns the MTTD each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getMTTDStats`**:

Request example

A cURL request to retrieve the MTTD for Moogsoft AIOps from 11.09am on Sunday 17th December until 11.09am on Sunday 24th December 2017:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getMTTDStats" --data-
urlencode 'from=1513508950' --data-urlencode 'to=1514113750'
```

Response example

Successful request returns the MTTD for the 24 hour time frame:

```
[{
        "datapoints": [
                [272.0, 1514113750000],
        ],
        "target": "Mean Time to Detect (MTTD)"
}]
```

getMTTRPerTeamStats

A GET request that returns the mean time to resolve (MTTR) a Situation per team for a given time range.

Back to Stats API.

Request arguments

Endpoint **`getMTTRPerTeamStats`** takes the following request arguments.

Name	Type	Description

auth_token	String	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
teams	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
aggregation	String	Set to: **accumulate**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getMTTRPerTeamStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Mean Time to Resolve (MTTR)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point:MTTR (seconds) for that bucket. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the MTTR each hour in the time period.1 week to 1 month: Returns the MTTR each day in the time period.1 month to 1 year: Returns the MTTR each week in the time period.More than 1 year: Returns the MTTR each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getMTTRPerTeamStats**:

Request example

A cURL request for the MTTR of the Cloud DevOps team from 9.26pm on Monday, November 6th until 2.26am on Tuesday, November 7th 2017:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getMTTRPerTeamStats" --
data-urlencode 'teams=[1]'  --data-urlencode 'from=1510003600' --data-urlencode
'to=1510021600' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the MTTR each hour from 9.26pm until 2.26am:

```
[{
        "datapoints": [
                [101.6,1510003600000],
                [180.0,1510007200000],
                [210.6667,1510010800000],
                [85.7083,1510014400000],
                [302.5,1510018000000],
                [150.4286,1510021600000]]
        ],
        "target": "Mean Time to Resolve (MTTR)"
}]
```

getMTTRPerUserStats

A GET request that returns the mean time it takes a user to resolve a Situation within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getMTTRPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getMTTRPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Mean Time to Resolve (MTTR) (full name)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]:

		Data point: MTTR (seconds) for that bucket. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of MTTR each hour in the time period. 1 week to 1 month: Returns the number of MTTR each day in the time period. 1 month to 1 year: Returns the number of MTTR each week in the time period. More than 1 year: Returns the number of MTTR each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getMTTRPerUserStats`**:

Request example

A cURL request for the MTTR for user 5 from 11pm on Monday, 1st October until 5am on Tuesday, 2nd October 2018:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getMTTRPerUserStats" --
data-urlencode 'user=[5]' --data-urlencode 'from=1538434800' --data-urlencode
'to=1538456400' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the MTTR each hour:

```
[{
        "datapoints": [
                [12997.0,1538434800000],
                [14025.0,1538438400000],
                [2969.0,1538442000000],
                [13125.0,1538445600000],
                [11412.0,1538449200000],
                [8264.0,1538452800000]
        ],
        "target": "Mean Time to Resolve (MTTR)(Oscar O'Neill)"
}]
```

getMTTRStats

A GET request that returns the Mean Time To Resolve (MTTR) for Situations in the specified time range.

The TTR for a Situation is the duration from the first event in the Situation to the time when a user resolved the Situation.

Back to Stats API.

Request arguments

Endpoint **`getMTTRStats`** takes the following request arguments.

Name	Type	Description

auth_token	String	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

Response

Endpoint **getMTTRStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Mean Time to Resolve (MTTR)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: MTTR (seconds) for that bucket Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the MTTR each hour in the time period.1 week to 1 month: Returns the MTTR each day in the time period.1 month to 1 year: Returns the MTTR each week in the time period.More than 1 year: Returns the MTTR each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getMTTRStats**:

Request example

A cURL request to retrieve the MTTR for Moogsoft AIOps from 11.30am on Sunday, September 24th 2017 until 11.30am on Sunday, September24th 2018:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getMTTRStats" --data-
urlencode 'from=1506252610' --data-urlencode 'to=1537788610'
```

Response example

A successful response indicates the MTTR for the year was 2.72 minutes:

```
[{
        "datapoints": [
                [163.54,1537784877233]
        ],
        "target":"Mean Time to Resolve (MTTR)"
}]
```

getNewAlertsPerSituationsStats

A GET request that returns the percentage of noise reduction from alerts-to-Situations clustering in the specified time range.

Back to Stats API.

Request arguments

Endpoint **getNewAlertsPerSituationsStats** takes the following request arguments.

Name	Type	Description
auth_token	String	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

Response

Endpoint **getNewAlertsPerSituationsStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"New Alerts per Situation"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Percentage noise reduction (alert to Situation reduction). Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the percentage noise reduction each hour in the time period.1 week to 1 month: Returns the percentage noise reduction each day in the time period.1 month to 1 year: Returns the percentage noise reduction each week in the time period.More than 1 year: Returns the percentage noise reduction each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getNewAlertsPerSituationsStats**:

Request example

Example cURL request to to retrieve the percentage noise reduction from 7.07pm on Wednesday, 17th January 2018 until 1.33pm on Thursday, 18th January 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getNewAlertsPerSituationsStats" --data-urlencode
'from=1516216020' --data-urlencode 'to=1516282420'
```

Response example

Example response indicating a noise reduction of 78.5% in the number of alerts to Situations:

```
[
        {"datapoints":[
                [78.5,1523438216685]
        ],
        "target":"New Alerts per Situation"}
]
```

getNewAlertsStats

A GET request that returns the number of new alerts in the specified time range.

Back to Stats API.

Request arguments

Endpoint **`getNewAlertsStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getNewAlertsStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"New Alerts"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of alerts Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of alerts each hour in the time period.1 week to 1 month: Returns the number of alerts each day in the time period.1 month to 1 year: Returns the number of alerts each week in the time period.More than 1 year: Returns the number of alerts each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getNewAlertsStats`**:

Request example

Example cURL request to retrieve the number of new alerts between Wednesday, January 17th and Thursday, January 18th 2018:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getNewAlertsStats" --
data-urlencode 'from=1516216020' --data-urlencode 'to=1516282420'
```

Response example

Example response that indicates there were 28,542 new alerts over the 24 hour time period: :

```
[
        {"datapoints":[
                [28542.0,1523438216685]
        ],
        "target":"New Alerts"}
]
```

getNewEventsPerAlertsStats

A GET request that returns the percentage of noise reduction from events-to-alerts aggregation and deduplication in the specified time range.

Back to Stats API.

Request arguments

Endpoint **`getNewEventsPerAlertsStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

Response

Endpoint **`getNewEventsPerAlertsStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"New Events per Alerts"

datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Percentage noise reduction (event to alert reduction). Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of percentage noise reduction each hour in the time period. 1 week to 1 month: Returns the number of percentage noise reduction each day in the time period. 1 month to 1 year: Returns the number of percentage noise reduction each week in the time period. More than 1 year: Returns the number of percentage noise reduction each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getNewEventsPerAlertsStats**:

Request example

A cURL request that retrieves that event to alert noise reduction in Moogsoft AIOps from 7.07pm on Wednesday, 17th January until 7.07pm on Thursday, 18th January 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getNewEventsPerAlertsStats" --data-urlencode
'from=1516216020' --data-urlencode 'to=1516302431'
```

Response example

A successful response indicating a 58% noise reduction:

```
[
        {"datapoints":[
                [58.0,1523438216685]
        ],
        "target":"New Events per Alerts"}
]
```

getNewEventsPerSituationsStats

A GET request that returns the percentage of noise reduction from events-to-Situations aggregation, deduplication, and clustering in the specified time range.

Back to Stats API.

Request arguments

Endpoint **getNewEventsPerSituationsStats** takes the following request arguments.

Name	Type	Description
auth_token	String	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.
from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint

		returned is the current state datapoint.

Response

Endpoint **`getNewEventsPerSituationsStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"New Events per Situation"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Percentage noise reduction (event to Situation reduction). Timestamp: Calculation time (Unix epoch timestamp in milliseconds). [Delete all except the appropriate Time Period box or complete the custom list if not supplied. Delete this para!] Less than 1 week: Returns the percentage noise reduction each hour in the time period.1 week to 1 month: Returns the percentage noise reduction each day in the time period.1 month to 1 year: Returns the percentage noise reduction each week in the time period.More than 1 year: Returns the percentage noise reduction each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getNewEventsPerSituationsStats`**:

Request example

A cURL request that retrieves the percentage noise reduction for the past month ranging from 10.28am on Sunday, August 26th until 10.28am on Wednesday, September 26th 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getNewEventsPerSituationsStats" --data-urlencode
'from=1533103200' --data-urlencode 'to=1535695200'
```

Response example

A successful responses returns an 95% to 96% reduction in events to Situations for each week over the past month:

```
[
        {
        "datapoints":[
                [95.86151338591529,1535279280000],
                [95.79150698161867,1535884080000],
                [95.62050414072417,1536488880000],
                [96.08938014241262,1537093680000],
                [95.96508799542137,1537698480000]
```

```
            ],
            "target":"New Events per Situation"
            }
    ]
```

getNewSituationsStats

A GET request that returns the number of new Situations created in the specified time range.

Back to Stats API.

Request arguments

Endpoint **`getNewSituationsStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

Response

Endpoint **`getNewSituationsStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"New Situations"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of new Situations. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of new Situations each hour in the time period. 1 week to 1 month: Returns the number of new Situations each day in the time period. 1 month to 1 year: Returns the number of new Situations each week in the time period. More than 1 year: Returns the number of new Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getNewSituationsStats`**:

Request example

Example cURL request to retrieve the number of new Situations over a week from 6am on Saturday, September 1st until 6am on Saturday, September 8th 2018:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getNewSituationsStats"
--data-urlencode 'from=1535781600' --data-urlencode 'to=1536386400'
```

Response example

Example response returning the number of new Situations for each day during the week range:

```
[
        {"datapoints":[
                [601.0,1535781600000],
                [523.0,1535868000000],
                [597.0,1535954400000],
                [618.0,1536040800000],
                [535.0,1536127200000],
                [628.0,1536213600000],
                [618.0,1536300000000]
        ],
        "target":"New situations"}
]
```

getOpenSituationsPerUserStats

A GET request that returns the number of open Situations assigned to a user at each data point.

Back to Stats API.

Request arguments

Endpoint **`getOpenSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getOpenSituationsPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Open Situations (full name)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of open Situations assigned to the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations assigned each hour in the time period. 1 week to 1 month: Returns the number of Situations assigned each day in the time period. 1 month to 1 year: Returns the number of Situations assigned each week in the time period. More than 1 year: Returns the number of Situations assigned each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getOpenSituationsPerUserStats**:

Request example

A cURL request to return the number of open Situations assigned to user 6 from 9.19am on Monday, 17th September until 16.19am on Monday, 17th September 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getOpenSituationsPerUserStats" --data-urlencode
'users=[6,7]' --data-urlencode 'from=1537175946' --data-urlencode
'to=1537201140' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of open Situations assigned to the users Oscar and Olivia each hour during the time range:

```
[{
        "datapoints":[
                [12.0,1537175946000],
                [8.875,1537262346000],
                [10.0,1537348746000],
                [8.9,1537435146000],
                [10.75,1537521546000],
                [9.25,1537607946000],
                [8.1667,1537694346000]
        ],
        "target":"Open Situations (Oscar O'Neill)"
},
{
        "datapoints":[
```

```
                [4.0,1537175946000],
                [5.0,1537262346000],
                [12.0,1537348746000],
                [7.0,1537435146000],
                [3.0,1537521546000],
                [9.0,1537607946000],
                [8.0,1537694346000]
            ],
            "target":"Open Situations (Andrew Anderson)"
    }
    ]
```

getRatedSituationsPerUserStats

A GET request that returns the number of Situations rated by a user within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getRatedSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getRatedSituationsPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Rated Situations (full name)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]:

		Data point: Number of Situations rated by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations rated each hour in the time period. 1 week to 1 month: Returns the number of Situations rated each day in the time period. 1 month to 1 year: Returns the number of Situations rated each week in the time period. More than 1 year: Returns the number of Situations rated each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getRatedSituationsPerUserStats`**:

Request example

A cURL request to return the number of Situations rated by users 5 and 7 from 3:57am until 9:57am on Thursday, October 5th 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getRatedSituationsPerUserStats" --data-urlencode
'users=[5,7]' --data-urlencode 'from=1538621843' --data-urlencode
'to=1538643443'
```

Response example

A successful response returns the number of Situations rated by the users Steve and Charlie each hour during the time range:

```
[{
        "datapoints":[
                [6.0,1538621843000],
                [1.0,1538625443000],
                [6.0,1538629043000],
                [5.0,1538632643000],
                [2.0,1538636243000],
                [5.0,1538639843000]
        ],
        "target":"Rated Situations (Steve Smith)"
  },
  {
        "datapoints":[
                [0.0,1538621843000],
                [3.0,1538625443000],
                [1.0,1538629043000],
                [6.0,1538632643000],
                [6.0,1538636243000],
                [8.0,1538639843000]
        ],
        "target":"Rated Situations (Charlie Copper)"
}]
```

getReassignedSituationsPerTeamStats

A GET request that returns the number of reassigned Situations associated with a team or multiple teams over a given time range. A reassigned Situation is a Situation that a user has assigned to another user at least twice.

Back to Stats API.

Request arguments

Endpoint **`getReassignedSituationsPerTeamStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`teams`	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getReassignedSituationsPerTeamStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	The name of the team: "<team_name>"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of reassigned Situations. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations reassigned each hour in the time period.1 week to 1 month: Returns the number of Situations reassigned each day in the time period.1 month to 1 year: Returns the number of Situations reassigned each week in the time period.More than 1 year: Returns the number of Situations reassigned each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getReassignedSituationsPerTeamStats`**:

Request example

A cURL request to retrieve the reassigned Situations for the Cloud DevOps and Application Performance Monitoring teams from August 1st until September 1st 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getReassignedSituationsPerTeamStats" --data-
urlencode 'teams=[1,2]' --data-urlencode 'from=1533103200' --data-urlencode
'to=1535781600'
```

Response example

A successful response returns the number of reassigned Situations for each week during that month range for both teams:

```
[{
    "datapoints":[
                [4.9702,1533103200000],
                [4.9881,1533708000000],
                [5.0655,1534312800000],
                [4.9524,1534917600000],
                [4.9917,1535522400000]],
    "target":"Cloud DevOps"
 },
 {
    "datapoints":[
                [5.006,1533103200000],
                [5.0,1533708000000],
                [5.131,1534312800000],
                [5.0714,1534917600000],
                [4.8417,1535522400000]],
    "target":"Application Performance Monitoring"
 }]
```

getReassignedSituationsPerUserStats

A GET request that returns the number of Situations reassigned by a user within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getReassignedSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

aggregation	String	Set to: **accumulate**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getReassignedSituationsPerUserStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Reassigned Situations (full name)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations reassigned by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations reassigned each hour in the time period. 1 week to 1 month: Returns the number of Situations reassigned each day in the time period. 1 month to 1 year: Returns the number of Situations reassigned each week in the time period. More than 1 year: Returns the number of Situations reassigned each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getReassignedSituationsPerUserStats**:

Request example

A cURL request to return the number of Situations reassigned by user 5 from 11pm on Sunday, 14th October until 5am on Monday, 15th October 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getReassignedSituationsPerUserStats" --data-
urlencode 'users=[5]' --data-urlencode 'from=1539558000' --data-urlencode
'to=1539579600' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of Situations reassigned by the user Dave each hour during the time range:

```
[{
        "datapoints":[
                [2.0,1539558000000],
                [3.0,1539561600000],
                [0.0,1539565200000],
                [1.0,1539568800000],
                [0.0,1539572400000],
                [3.0,1539576000000],
        ],
        "target":"Reassigned Situations (Dave Danton)"
}]
```

getReassignedSituationStats

A GET request that returns the number of Situations reassigned in the specified time range. A reassigned Situation is a Situation that a user has assigned to another user at least twice.

Back to Stats API.

Request arguments

Endpoint **`getReassignedSituationStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

Response

Endpoint **`getReassignedSituationStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Reassigned Situation"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of reassigned Situations. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations reassigned each hour in the time period.1 week to 1 month: Returns the number of Situations reassigned each day in the time period.1 month to 1 year: Returns the number of Situations reassigned each week in the time period.More than 1 year: Returns the number of Situations reassigned each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getReassignedSituationStats`**:

Request example

A cURL request to retrieve the number of reassigned Situations over a month from 6am on Wednesday, August 1st until 6am on Saturday, September 1st 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getReassignedSituationStats" --data-urlencode
'from=1533103200' --data-urlencode 'to=1535781600'
```

Response example

A successful response returns the number of reassigned Situations for each week during the month:

```
[{
        "datapoints": [
                [25.125,1533103200000],
                [24.1369,1533708000000],
                [25.9405,1534312800000],
                [24.8512,1534917600000],
                [25.1071,1535522400000],
        ],
        "target": "Reassigned Situation"
}]
```

getReoccurringSituationPerTeamStats

A GET request that returns the number of reoccurring Situations associated with a team for a given time range.

Back to Stats API.

Request arguments

Endpoint **`getReoccurringSituationPerTeamStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`teams`	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **getReoccurringSituationPerTeamStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Reoccurring situations"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of reoccurring Situations. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations reoccurring each hour in the time period.1 week to 1 month: Returns the number of Situations reoccurring each day in the time period.1 month to 1 year: Returns the number of Situations reoccurring each week in the time period.More than 1 year: Returns the number of Situations reoccurring each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getReoccurringSituationPerTeamStats**:

Request example

A cURL request to retrieve the number of reoccuring Situations from 3pm on Saturday, September 1st until 3pm on Saturday, September 8th 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getReoccurringSituationPerTeamStats" --data-
urlencode 'teams=[1,2]' --data-urlencode 'from=1535814000' --data-urlencode
'to=1536418800' --data-urlencode 'aggregation=none'
```

Response example

A successful response indicates there were four reoccuring Situations at the time the request was sent:

```
[{"datapoints":[[4.0,1538044321144]],"target":"Reoccurring situations"}]
```

getReoccurringSituationStats

A GET request that returns the percentage of reoccurring situations in the system over a given time range.

Back to Stats API.

Request arguments

Endpoint **getReoccurringSituationStats** takes the following request arguments.

Name	Type	Description
auth_token	String	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

Response

Endpoint **getReoccurringSituationStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	"Reoccurring Situations"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of reoccurring Situations. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations reoccurring each hour in the time period.1 week to 1 month: Returns the number of Situations reoccurring each day in the time period.1 month to 1 year: Returns the number of Situations reoccurring each week in the time period.More than 1 year: Returns the number of Situations reoccurring each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getReoccurringSituationStats**:

Request example

A cURL request to retrieve the number of reoccurring Situations from 6pm on Sunday, September 10th 2017 until 6pm on Monday, September 10th 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getReoccurringSituationStats" --data-urlencode
'from=1505066400' --data-urlencode 'to=1536602400'
```

Response example

A successful response returns that there were 186 reoccurring Situations during the year:

```
[{
        "datapoints": [
                [186.0, 1537980650126],
        ],
        "target": "Reoccurring situations"
}]
```

getResolvedSituationsPerUserStats

A GET request that returns the number of Situations resolved by a user within a given time range.

Back to Stats API.

Request arguments

Endpoint **`getResolvedSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getResolvedSituationsPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Resolved Situations (full name)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations resolved by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations resolved each hour in the time period. 1 week to 1 month: Returns the number of Situations resolved each day in the time period. 1 month to 1 year: Returns the number of Situations resolved each week in the time period. More than 1 year: Returns the number of Situations resolved each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getResolvedSituationsPerUserStats`**:

Request example

A cURL request to return the number of Situations resolved by user 5 from 8.47am until 15.04pm on October 1st 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getResolvedSituationsPerUserStats" --data-urlencode
'users=[5]' --data-urlencode 'from=1538380070' --data-urlencode 'to=1538402670'
--data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of Situations resolved by the user Alice each hour during the time range:

```
[{
        "datapoints":[
                [5.0,1538380070000],
                [3.0,1538383670000],
                [8.0,1538387270000],
                [0.0,1538390870000],
                [0.0,1538394470000],
                [8.0,1538398070000],
        ],
        "target":"Resolved Situations (Alice Anderson)"
}]
```

getServiceSituationPerTeamStats

A GET request that returns the number of Situations impacting each service for a team.

Back to Stats API.

Request arguments

Endpoint **`getServiceSituationPerTeamStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`teams`	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to:

		accumulate: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getServiceSituationPerTeamStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	The name of the team.
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations impacting services. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getServiceSituationPerTeamStats**:

Request example

A cURL request to retrieve the number of Situations associated with the Cloud DevOps team that are impacting the Commerce and Compute services between 12pm and 6pm on Friday, August 10th 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getServiceSituationPerTeamStats" --data-urlencode
'from=1533902400' --data-urlencode 'to=1533924000' --data-urlencode 'teams=[1]'
--data-urlencode 'services=[1, 2]' --data-urlencode 'aggregation=none'
```

Response example

A successful request returns the number of Situations impacting the services each hour during the six hour time range:

```
[{
        "datapoints":[
                [7.0,1533902400000],
                [18.0,1533906000000],
                [18.0,1533909600000],
                [13.0,1533913200000],
                [9.0,1533916800000],
                [12.0,1533920400000]],
        "target":"Commerce"},
{
        "datapoints":[
                [14.0,1533902400000],
```

```
                [15.0,1533906000000],
                [6.0,1533909600000],
                [12.0,1533913200000],
                [1.0,1533916800000],
                [11.0,1533920400000]],
        "target":"Compute"
}]
```

getServiceSituationStats

A GET request that returns the number of active Situations impacting a service in the specified time range.

Back to Stats API.

Request arguments

Endpoint **`getServiceSituationStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`services`	Array	An array of services IDs. If no services are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getServiceSituationStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	Service name(s).
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations impacting services.

		Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getServiceSituationStats`** :

Request example

A cURL request to retrieve the number of Situations impacting the Commerce/Compute service between 12pm and 6pm on Friday, August 10th 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getServiceSituationStats" --data-urlencode
'services=[1,2]' --data-urlencode 'from=1533902400' --data-urlencode
'to=1533924000' --data-urlencode 'aggregation=sum'
```

Response example

A successful response returns six data points for each hour during the six hour time range:

```
[{
        "datapoints": [
                [95.0,1533902400000],
                [85.0,1533906000000],
                [47.0,1533909600000],
                [7.0,1533913200000],
                [33.0,1533916800000],
                [66.0,1533920400000]
        ],
        "target":"Commerce/Compute"
}]
```

getSeveritySituationPerTeamStats

A GET request that returns the number of Situations by severity per team for a given time range.

Back to Stats API.

Request arguments

Endpoint **`getSeveritySituationPerTeamStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`teams`	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

aggregation	String	Set to: **accumulate**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getSeveritySituationPerTeamStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	The name of the status.
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations per severity. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getSeveritySituationPerTeamStats**:

Request example

A cURL request to retrieve the number of clear Situations for the Cloud DevOps team between between 12pm onThursday, August 9th and 12pm on Friday, August 10th 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getSeveritySituationPerTeamStats" --data-urlencode
'from=1533816000' --data-urlencode 'to=1533902400' --data-urlencode 'teams=[1]'
--data-urlencode 'severity=[0]' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of clear Situations each hour over the past 24 hours:

```
[{
        "datapoints":[
                [13.0,1533816000000],
                [14.0,1533819600000],
                [6.0,1533823200000],
                [10.0,1533826800000],
                [14.0,1533830400000],
```

```
                [5.0,1533834000000],
                [19.0,1533837600000],
                [17.0,1533841200000],
                [4.0,1533844800000],
                [13.0,1533848400000],
                [7.0,1533852000000],
                [15.0,1533855600000],
                [6.0,1533859200000],
                [10.0,1533862800000],
                [16.0,1533866400000],
                [20.0,1533870000000],
                [19.0,1533873600000],
                [15.0,1533877200000],
                [15.0,1533880800000],
                [5.0,1533884400000],
                [20.0,1533888000000],
                [3.0,1533891600000],
                [1.0,1533895200000],
                [4.0,1533898800000]],
        "target":"Clear"
}]
```

getSeveritySituationStats

A GET request that returns the number of Situations by severity in the specified time range.

Back to Stats API.

Request arguments

Endpoint **`getSeveritySituationStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`services`	Array	An array of services IDs. If no services are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getSeveritySituationStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	The name of the status.
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations per severity. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getSeveritySituationStats**:

Request example

A cURL request to retrieve the sum of the major and critical Situations between 12pm on Thursday, August 9th and 12pm on Friday, August 10th 2018:

```
curl -G -u graze:graze -k -v
"https://daffy.moogsoft.com/graze/v1/getSeveritySituationStats" --data-urlencode
'from=1533816000' --data-urlencode 'to=1533902400' --data-urlencode
'severity=[5, 4]' --data-urlencode 'aggregation=sum'
```

Response example

A successful response returns 24 data points, one for each hour over the 24 hour range:

```
[{
        "datapoints":[
                [51.0,1533816000000],
                [44.0,1533819600000],
                [88.0,1533823200000],
                [84.0,1533826800000],
                [25.0,1533830400000],
                [34.0,1533834000000],
                [82.0,1533837600000],
                [58.0,1533841200000],
                [61.0,1533844800000],
                [52.0,1533848400000],
                [15.0,1533852000000],
                [50.0,1533855600000],
                [54.0,1533859200000],
                [50.0,1533862800000],
                [81.0,1533866400000],
                [78.0,1533870000000],
                [84.0,1533873600000],
                [28.0,1533877200000],
                [54.0,1533880800000],
                [36.0,1533884400000],
                [44.0,1533888000000],
```

```
                [47.0,1533891600000],
                [60.0,1533895200000],
                [54.0,1533898800000]],
          "target":"Critical/Major"
}]
```

getStats

A GET request that retrieves all available Stats API endpoints along with their description and request parameters.

Back to Stats API.

Request arguments

Name	Type	Required	Description
auth_token	String	Yes	A valid **auth_token** returned from the **authenticate** request. See the authenticate endpoint for more information.

Endpoint **getStats** takes no other arguments because it returns data on all available Stats API endpoints.

Response

Endpoint **getStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Examples

The following examples demonstrate typical use of endpoint **getStats**:

Request example

A cURL request to return all available Stats API endpoints:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getStats"
```

Response example

A successful response with all of the endpoints, descriptions and associated parameters:

```
[
 {
   "endpoint":"getTeamSituationStats",
   "description":"returns the number of active situations assign to a team over
time",
   "display_name":"Open Situations by Team",
   "parameters":{
      "teams":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getTeams",
            "value":"team_id"
         },
         "type":"mapped",
         "required":false
      },
      "from":{
         "description":"A timestamp from epoch in seconds",
```

```
            "type":"Long",
            "required":true
         },
         "aggregation":{
            "default":"none",
            "type":"string",
            "static_mapping":[
               {
                  "display_value":"None",
                  "value":"none"
               },
               {
                  "display_value":"Sum",
                  "value":"sum"
               }
            ],
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         }
      }
   },
   {
      "endpoint":"getTopTeamSituationStats",
      "description":"returns the number of active situations assign to a top team
   over time",
      "display_name":"Open Situations by Top Team",
      "parameters":{
         "from":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "aggregation":{
            "default":"none",
            "type":"string",
            "static_mapping":[
               {
                  "display_value":"None",
                  "value":"none"
               },
               {
                  "display_value":"Sum",
                  "value":"sum"
               }
            ],
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         }
      }
```

```
},
{
   "endpoint":"getServiceSituationStats",
   "description":"returns the number of active situations impacting a service
over time",
   "display_name":"Open Situations by Service",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Sum",
               "value":"sum"
            }
         ],
         "required":false
      },
      "services":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getServices",
            "value":"service_id"
         },
         "type":"mapped",
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getTopServiceSituationStats",
   "description":"returns the number of active situations impacting a top
service over time",
   "display_name":"Open Situations by Top Service",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
```

```
            },
            {
               "display_value":"Sum",
               "value":"sum"
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getSystemSituationStats",
   "description":"returns the number of active situations in the system over
time",
   "display_name":"All Open Situations",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getStatusSituationStats",
   "description":"returns the number of active situations with specified status
over time",
   "display_name":"Open Situations by Status",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Sum",
               "value":"sum"
            }
         ],
```

```
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "status":{
            "mapping":{
               "display_value":"name",
               "endpoint":"getStatuses",
               "value":"status_id"
            },
            "type":"mapped",
            "required":false
         }
      }
   },
   {
      "endpoint":"getSeveritySituationStats",
      "description":"returns the number of active situations with specified
   severity over time",
      "display_name":"Open Situations by Severity",
      "parameters":{
         "severity":{
            "mapping":{
               "display_value":"name",
               "endpoint":"getSeverities",
               "value":"severity_id"
            },
            "type":"mapped",
            "required":"false"
         },
         "from":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "aggregation":{
            "default":"none",
            "type":"string",
            "static_mapping":[
               {
                  "display_value":"None",
                  "value":"none"
               },
               {
                  "display_value":"Sum",
                  "value":"sum"
               }
            ],
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         }
      }
   },
   {
```

```
   "endpoint":"getReoccurringSituationStats",
   "description":"returns the percentage of reoccurring situations in the
system",
   "display_name":"Reoccurring situations",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getMTTAStats",
   "description":"returns the mean time to acknowledge a situation over time",
   "display_name":"Mean Time To Acknowledge",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getMTTDStats",
   "description":"returns the mean time to detect a situation over time",
   "display_name":"Mean Time To Detect",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getMTTRStats",
   "description":"returns the mean time to resolve a situation over time",
   "display_name":"Mean Time To Resolve",
   "parameters":{
      "from":{
```

```
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         }
      }
   },
   {
      "endpoint":"getReassignedSituationStats",
      "description":"returns the number of situations that have been reassigned
   over time",
      "display_name":"Reassigned Situations",
      "parameters":{
         "from":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         }
      }
   },
   {
      "endpoint":"getNewSituationsStats",
      "description":"returns the number of new situations over time",
      "display_name":"New Situations",
      "parameters":{
         "from":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "aggregation":{
            "default":"none",
            "type":"string",
            "static_mapping":[
               {
                  "display_value":"None",
                  "value":"none"
               },
               {
                  "display_value":"Accumulate",
                  "value":"accumulate"
               }
            ],
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         }
      }
   },
```

```
{
   "endpoint":"getNewAlertsStats",
   "description":"returns the number of new alerts over time",
   "display_name":"New Alerts",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Accumulate",
               "value":"accumulate"
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getNewEventsStats",
   "description":"returns the number of new events over time",
   "display_name":"New Events",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Accumulate",
               "value":"accumulate"
            }
         ],
         "required":false
      },
```

```
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getAlertsInNewSituationsStats",
   "description":"returns the number of alerts in new situations over time",
   "display_name":"Alerts In New Situations",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Accumulate",
               "value":"accumulate"
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getNewEventsPerAlertsStats",
   "description":"returns the number of new events divided by the number of new
alerts over time",
   "display_name":"Reduction From Events To Alert",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Accumulate",
               "value":"accumulate"
```

```
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getNewAlertsPerSituationsStats",
   "description":"returns the number of new alerts divided by the number of new
situations over time",
   "display_name":"Reduction From Alerts To Situations",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Accumulate",
               "value":"accumulate"
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getNewEventsPerSituationsStats",
   "description":"returns the number of new events divided by the number of new
situations over time",
   "display_name":"Reduction From Events To Situations",
   "parameters":{
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
```

```
            "type":"string",
            "static_mapping":[
               {
                  "display_value":"None",
                  "value":"none"
               },
               {
                  "display_value":"Accumulate",
                  "value":"accumulate"
               }
            ],
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         }
      }
   },
   {
      "endpoint":"getReassignedSituationsPerTeamStats",
      "description":"returns the number of reassigned situations of a team over
   time",
      "display_name":"Reassigned Situations by Team",
      "parameters":{
         "teams":{
            "mapping":{
               "display_value":"name",
               "endpoint":"getTeams",
               "value":"team_id"
            },
            "type":"mapped",
            "required":false
         },
         "from":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "aggregation":{
            "default":"none",
            "type":"string",
            "static_mapping":[
               {
                  "display_value":"None",
                  "value":"none"
               },
               {
                  "display_value":"Sum",
                  "value":"sum"
               }
            ],
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         }
      }
```

```
},
{
   "endpoint":"getSeveritySituationPerTeamStats",
   "description":"returns the number of active situations with specified
severity and team over time",
   "display_name":"Open Situations by Severity by Team",
   "parameters":{
      "severity":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getSeverities",
            "value":"severity_id"
         },
         "type":"mapped",
         "required":"false"
      },
      "teams":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getTeams",
            "value":"team_id"
         },
         "type":"mapped",
         "required":false
      },
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Sum",
               "value":"sum"
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getStatusSituationPerTeamStats",
   "description":"returns the number of situations with a specified status and
team over time",
   "display_name":"Open Situations by Status by Team",
```

```
      "parameters":{
         "teams":{
            "mapping":{
               "display_value":"name",
               "endpoint":"getTeams",
               "value":"team_id"
            },
            "type":"mapped",
            "required":false
         },
         "from":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "aggregation":{
            "default":"none",
            "type":"string",
            "static_mapping":[
               {
                  "display_value":"None",
                  "value":"none"
               },
               {
                  "display_value":"Sum",
                  "value":"sum"
               }
            ],
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "status":{
            "mapping":{
               "display_value":"name",
               "endpoint":"getStatuses",
               "value":"status_id"
            },
            "type":"mapped",
            "required":"false"
         }
      }
   },
   {
      "endpoint":"getServiceSituationPerTeamStats",
      "description":"returns the number of active situations with specified service
   and team over time",
      "display_name":"Open Situations by Service by Team",
      "parameters":{
         "teams":{
            "mapping":{
               "display_value":"name",
               "endpoint":"getTeams",
               "value":"team_id"
            },
            "type":"mapped",
            "required":true
         },
```

```
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Sum",
               "value":"sum"
            }
         ],
         "required":false
      },
      "services":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getServices",
            "va* Connection #0 to host freida7 left intact
lue":"service_id"
         },
         "type":"mapped",
         "required":"true"
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getMTTAPerTeamStats",
   "description":"returns the mean time to acknowledge a situation of a team
over time",
   "display_name":"Mean Time To Acknowledge by Team",
   "parameters":{
      "teams":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getTeams",
            "value":"team_id"
         },
         "type":"mapped",
         "required":false
      },
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
```

```
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Sum",
               "value":"sum"
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getMTTRPerTeamStats",
   "description":"returns the mean time to resolve a situation of a team over
time",
   "display_name":"Mean Time To Resolve by Team",
   "parameters":{
      "teams":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getTeams",
            "value":"team_id"
         },
         "type":"mapped",
         "required":false
      },
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Sum",
               "value":"sum"
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
```

```
      }
   }
},
{
   "endpoint":"getReoccurringSituationPerTeamStats",
   "description":"returns the percentage of reoccurring situations of a team
over time",
   "display_name":"Reoccurring situations Per Team",
   "parameters":{
      "teams":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getTeams",
            "value":"team_id"
         },
         "type":"mapped",
         "required":false
      },
      "from":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      },
      "aggregation":{
         "default":"none",
         "type":"string",
         "static_mapping":[
            {
               "display_value":"None",
               "value":"none"
            },
            {
               "display_value":"Sum",
               "value":"sum"
            }
         ],
         "required":false
      },
      "to":{
         "description":"A timestamp from epoch in seconds",
         "type":"Long",
         "required":true
      }
   }
},
{
   "endpoint":"getCommentCountPerTeamStats",
   "description":"returns the number of comments posted on situations by team
members over time",
   "display_name":"Number of Comments by Team",
   "parameters":{
      "teams":{
         "mapping":{
            "display_value":"name",
            "endpoint":"getTeams",
            "value":"team_id"
         },
```

```
            "type":"mapped",
            "required":false
         },
         "from":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
         },
         "aggregation":{
            "default":"none",
            "type":"string",
            "static_mapping":[
               {
                  "display_value":"None",
                  "value":"none"
               },
               {
                  "display_value":"Sum",
                  "value":"sum"
               }
            ],
            "required":false
         },
         "to":{
            "description":"A timestamp from epoch in seconds",
            "type":"Long",
            "required":true
          }
        }
    }
]
```

getStatusSituationPerTeamStats

A GET request that returns the number of Situations by status for a team over a given time range.

Back to Stats API.

Request arguments

Endpoint **`getStatusSituationPerTeamStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`teams`	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getStatusSituationPerTeamStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	The name of the team.
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations for each status. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getStatusSituationPerTeamStats`**:

Request example

A cURL request to return all Situations by status for the Cloud DevOps team from 8.30am until 2.30pm on Saturday, September 1st 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getStatusSituationPerTeamStats" --data-urlencode
'from=1535790600' --data-urlencode 'to=1535812200' --data-urlencode 'teams=[1]'
--data-urlencode 'status=[]' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of Situations by status each hour for the six hour range:

```
[
        {"datapoints":[
                [19.0,1535790600000],
                [20.0,1535794200000],
                [17.0,1535797800000],
                [18.0,1535801400000],
                [17.0,1535805000000],
                [17.0,1535808600000]],
                "target":"Opened"},
        {"datapoints":[
                [3.0,1535790600000],
                [7.0,1535794200000],
                [4.0,1535797800000],
```

```
                [10.0,1535801400000],
                [10.0,1535805000000],
                [2.0,1535808600000]],
                "target":"Assigned"},
            {"datapoints":[
                [3.0,1535790600000],
                [5.0,1535794200000],
                [10.0,1535797800000],
                [3.0,1535801400000],
                [5.0,1535805000000],
                [2.0,1535808600000]],
                "target":"Acknowledged"},
            {"datapoints":[
                [3.0,1535790600000],
                [3.0,1535794200000],
                [4.0,1535797800000],
                [3.0,1535801400000],
                [3.0,1535805000000],
                [2.0,1535808600000]],
                "target":"Unacknowledged"},
            {"datapoints":[
                [46.0,1535790600000],
                [48.0,1535794200000],
                [32.0,1535797800000],
                [48.0,1535801400000],
                [34.0,1535805000000],
                [36.0,1535808600000]],
                "target":"Resolved"}
    ]
```


getStatusSituationStats

A GET request that returns the number of Situations by status.

Back to Stats API.

Request arguments

Endpoint **`getStatusSituationStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`status`	Array	An array of status ids. This is optional. If not given, it returns the default set of statuses: Opened, Unassigned, Assigned, Acknowledged, Unacknowledged, Resolved.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getStatusSituationStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	The status name.
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations for each status Timestamp: Calculation time (Unix epoch timestamp in milliseconds). [Delete all except the appropriate Time Period box or complete the custom list if not supplied. Delete this para!] Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getStatusSituationStats`**:

Request example

A cURL request to retrieve the number of opened and assigned Situations from 15.27pm on Sunday, January 14th until 15.27pm on Monday, 15th January 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getStatusSituationStats" --data-urlencode
'from=1515943678' --data-urlencode 'to=1516030078' --data-urlencode 'status=[1,
2]' --data-urlencode 'aggregation=sum'
```

Response example

Example response returning the number of Situations for each status: :

```
[{
        "datapoints": [
                [32.0, 1516008478000],
                [54.0, 1516030078000]
                [68.0, 1516030078000]
                [82.0, 1516030078000]
                [88.0, 1516030078000]
        ],
        "target": "Opened"
}, {
        "datapoints": [
                [5.0, 1515947278000],
                [12.0, 1515958078000],
```

```
                [25.0, 1515976078000],
                [31.0, 1515994078000],
                [40.0, 1516015678000]
            ],
            "target": "Assigned"
    }]
```

getSystemSituationStats

A GET request that returns the number of active Situations in the specified time range.

Back to Stats API.

Request arguments

Endpoint **`getSystemSituationStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.

Response

Endpoint **`getSystemSituationStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"System"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations for each Status Timestamp: Calculation time (Unix epoch timestamp in milliseconds). [Delete all except the appropriate Time Period box or complete the custom list if not supplied. Delete this para!] Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getSystemSituationStats`**:

Request example

A cURL request to retrieve the number of active Situations from 11.09am on Sunday, 17th December until 11.09am on Monday, 18th December 2017:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getSystemSituationStats" --data-urlencode
'from=1513508950' --data-urlencode 'to=1513595370'
```

Response example

A successful response returns the number of active Situations every hour during that time range:

```
[{
        "datapoints": [
                [66.0, 1513657700000],
                [98.0, 1513661300000],
                [102.0, 1513664900000],
                [106.0, 1513668500000],
                [92.0, 1513672100000],
                [88.0, 1513675700000],
                [86.0, 1513679300000],
                [74.0, 1513682900000],
                [85.0, 1513672100000],
                [83.0, 1513675700000],
                [79.0, 1513679300000],
                [68.0, 1513686500000]
        ],
        "target": "Open Situations"
}]
```

getTeamSituationStats

A GET request that returns the number of active Situations assigned to a team for a given time range.

Back to Stats API.

Request arguments

Endpoint **`getTeamSituationStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`teams`	Array	An array of team IDs. This is required. If no teams are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getTeamSituationStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	The name of the team.
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations for each status. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getTeamSituationStats`**:

Request example

A cURL request to return the number of active Situations assigned to the Cloud DevOps and Application Performance Monitoring teams from midnight until 6am on Monday, 20th August 2018:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getTeamSituationStats"
--data-urlencode 'teams=[1,2]' --data-urlencode 'from=1534723200' --data-
urlencode 'to=1534744800' --data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of Situations assigned each hour to each team for the six hour range:

```
[
        {"datapoints":[
                [30.0,1534723200000],
                [20.0,1534726800000],
                [24.0,1534730400000],
                [19.0,1534734000000],
                [28.0,1534737600000],
                [23.0,1534741200000]],
                "target":"Cloud DevOps"},
        {"datapoints":[
                [26.0,1534723200000],
                [29.0,1534726800000],
                [15.0,1534730400000],
                [29.0,1534734000000],
                [25.0,1534737600000],
                [22.0,1534741200000]],
                "target":"Application Performance Monitoring"]
```

getTopServiceSituationStats

A GET request that returns the number of active Situations impacting a top service in the specified time range. Top services are the services that have the most situations impacting them.

Back to Stats API.

Request arguments

Endpoint **`getTopServiceSituationStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getTopServiceSituationStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	The name of the service
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations for each status. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getTopServiceSituationStats`**:

Request example

A cURL request to retrieve the number of Situations impacting top services between 12pm and midnight on Saturday, 15th September 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getServiceSituationStats" --data-urlencode
'from=1537012800' --data-urlencode 'to=1536969600' --data-urlencode
'aggregation=sum'
```

Response example

A successful response returns the number of Situations each hour for the 12 hour range:

```
[{
        "datapoints": [
                [10.0, 1538133600000],
                [12.0, 1538133600000],
                [8.0, 1538133600000],
                [5.0, 1538133600000],
                [9.0, 1538133600000],
                [6.0, 1538133600000],
                [10.0, 1538133600000],
                [13.0, 1538133600000],
                [11.0, 1538133600000],
                [7.0, 1538133600000],
                [9.0, 1538133600000],
                [1.0, 1538133600000]
        ],
        "target": "Web Service"
}, {
        "datapoints": [
                [7.0, 1538133600000],
                [3.0, 1538133600000],
                [6.0, 1538133600000],
                [14.0, 1538133600000],
                [9.0, 1538133600000],
                [8.0, 1538133600000],
                [12.0, 1538133600000],
                [11.0, 1538133600000],
                [8.0, 1538133600000],
                [4.0, 1538133600000],
                [6.0, 1538133600000],
                [3.0, 1538133600000]],
        "target": "Cloud Service"
}]
```

getTopTeamSituationStats

A GET request that returns the number of active Situations assign to top teams over a given range of time. Top teams are those teams with the highest number of assigned Situations.

Back to Stats API.

Request arguments

Endpoint **`getTopTeamSituationStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.

from	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
to	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
aggregation	String	Set to: **accumulate**: Gradually adds data points together over time. **none**: No aggregation of data points.

Response

Endpoint **getTopTeamSituationStats** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
target	String	The name of the team.
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations for each status. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations each hour in the time period.1 week to 1 month: Returns the number of Situations each day in the time period.1 month to 1 year: Returns the number of Situations each week in the time period.More than 1 year: Returns the number of Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **getTopTeamSituationStats**:

Request example

A cURL request to retrieve the number of Situations impacting top teams between 6am and 12pm on Wednesday, 1st August 2018:

```
curl -G -u graze:graze -k -v "https://localhost/graze/v1/getTeamSituationStats"
--data-urlencode 'from=1533103200' --data-urlencode 'to=1533124800' --data-
urlencode 'aggregation=sum'
```

Response example

A successful response returns the number of Situations per hour for the six hour time time range:

```
[{
        "datapoints": [
```

```
                [2.0, 1538133780000],
                [9.0, 1538133780000],
                [5.0, 1538133780000],
                [4.0, 1538133780000],
                [3.0, 1538133780000],
                [1.0, 1538133780000]
        ],
        "target": "Cloud DevOps"
}, {
        "datapoints": [
                [8.0, 1538133780000],
                [2.0, 1538133780000],
                [6.0, 1538133780000],
                [7.0, 1538133780000],
                [5.0, 1538133780000],
                [3.0, 1538133780000]
        ],
        "target": "Application Performance Monitoring"
}]
```

getViewedSituationsPerUserStats

A GET request that returns the number of Situations a user has viewed within a given time range. Moogsoft AIOps considers a user to have viewed a Situation if they opened the Situation Room.

Back to Stats API.

Request arguments

Endpoint **`getViewedSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getViewedSituationsPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description

target	String	"Viewed Situations (full name)"
datapoints	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations viewed by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds). Less than 1 week: Returns the number of Situations viewed each hour in the time period. 1 week to 1 month: Returns the number of Situations viewed each day in the time period. 1 month to 1 year: Returns the number of Situations viewed each week in the time period. More than 1 year: Returns the number of Situations viewed each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getViewedSituationsPerUserStats`**:

Request example

A cURL request to return the number of viewed Situations by user 7 from 9am until 3pm on Thursday, 20th September 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getViewedSituationsPerUserStats" --data-urlencode
'users=[7]' --data-urlencode 'from=1537434000' --data-urlencode 'to=1537455600'
--data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of Situations viewed by the user Charlie each hour during the time range:

```
[{
        "datapoints":[
                [16.0,1537434000000],
                [26.0,1537437600000],
                [18.0,1537441200000],
                [34.0,1537444800000],
                [18.0,1537448400000],
                [11.0,1537452000000]
        ],
        "target":"Viewed Situations (Charlie Cooper)"
}]
```

getWorkedSituationsPerUserStats

A GET request that returns the number of Situations a user has worked on within a given time range.

Cisco Cisco Crosswork Situation Manager considers a user to have worked on a Situation if the user has:

1. Been assigned a Situation.
2. Been invited to a Situation.
3. Left a comment on a Situation.
4. Closed a Situation.
5. Resolved a Situation.
6. Executed a ChatOps tool on a Situation.
7. Rated a Situation.
8. Added PRC data to alerts in a Situation.

Back to Stats API.

Request arguments

Endpoint **`getWorkedSituationsPerUserStats`** takes the following request arguments.

Name	Type	Description
`auth_token`	String	A valid **`auth_token`** returned from the **`authenticate`** request. See the authenticate endpoint for more information.
`users`	Array	An array of user IDs. If no users are provided, the endpoint does not return any data.
`from`	Number	Start of the reporting time range. This is a Unix epoch timestamp in seconds.
`to`	Number	End of the reporting time range. This is a Unix epoch timestamp in seconds. If this timestamp is within 10 seconds of the current system time, the last datapoint returned is the current state datapoint.
`aggregation`	String	Set to: **`accumulate`**: Gradually adds data points together over time. **`none`**: No aggregation of data points.

Response

Endpoint **`getWorkedSituationsPerUserStats`** returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`target`	String	"Worked Situations (full name)"
`datapoints`	Number Array	An array of data points. Each data point is an array in the format [data point, timestamp]: Data point: Number of Situations worked on by the user. Timestamp: Calculation time (Unix epoch timestamp in milliseconds).

		Less than 1 week: Returns the number of worked Situations each hour in the time period. 1 week to 1 month: Returns the number of worked Situations each day in the time period. 1 month to 1 year: Returns the number of worked Situations each week in the time period. More than 1 year: Returns the number of worked Situations each month in the time period.

Examples

The following examples demonstrate typical use of endpoint **`getWorkedSituationsPerUserStats`**:

Request example

A cURL request to return the number of Situations worked on by user 5 from 12:22pm on Thursday 30th August until 8:22am Friday 31st August 2018:

```
curl -G -u graze:graze -k -v
"https://localhost/graze/v1/getWorkedSituationsPerUserStats" --data-urlencode
'users=[5]' --data-urlencode 'from=1535628143' --data-urlencode 'to=1535700143'
--data-urlencode 'aggregation=none'
```

Response example

A successful response returns the number of Situations worked by the user Chris each hour during the time range:

```
[{
        "datapoints":[
                [12.0,1535628143000],
                [25.0,1535631743000],
                [33.0,1535635343000],
                [14.0,1535638943000],
                [1.0,1535642543000],
                [4.0,1535646143000],
                [9.0,1535649743000],
                [6.0,1535653343000],
                [37.0,1535656943000],
                [31.0,1535660543000],
                [19.0,1535664143000],
                [35.0,1535667743000],
                [36.0,1535671343000],
                [28.0,1535674943000],
                [30.0,1535678543000],
                [19.0,1535682143000],
                [21.0,1535685743000],
                [30.0,1535689343000],
                [35.0,1535692943000],
                [30.0,1535696543000]
        ],
        "target":"Worked Situations (Chris Cole)"
}]
```

Integrations API

The Integrations API acts as an integration point for external services and exposes selected Cisco Crosswork Situation Manager functionality to authorized external clients.

Contact Cisco Support if you experience difficulties or need further guidance.

Endpoints

See Integrations API Endpoint Reference for details of all the Integrations API endpoints.

API definition

All Integrations requests use the following URL format, where **<server>** is the hostname of the machine running the Cisco Crosswork Situation Manager UI :

```
https://<server>/integrations/api/v1/<endpoint>
```

Examples:

```
https://example.com/integrations/api/v1/integrations
```

```
https://example.com/integrations/api/v1/integrations/{integrationId}
```

```
https://example.com/integrations/api/v1/integrations/{integrationId}/status
```

Authentication

In order to use the Integrations API, you must have the **manage_integrations** and **graze_login** permissions. The Grazer role has both of these permissions in Cisco Crosswork Situation Manager v8.x.

See Role Permissions for more information.

All requests require a basic authentication header.

Integrations API Endpoint Reference

This is a reference list for the Integrations API endpoints. Follow the links to see the details of each endpoint.

All of the endpoints use basic authorization.

Brokers

The following endpoints relate to brokers:

1. /broker-profiles: Create broker profiles.Create a Broker Profile

Integrations

The following endpoints relate to integration management:

1. /integrations: Create and configure new integrations.
2. /integrations/{integrationId}: Retrieve and update existing integrations.
3. /integrations/{integrationId}/status: Retrieve the status of an integration.

Workflows

The following example workflows combine queries to configure integrations:

1. Export and Import Integrations

2. Manage Integration States

/integrations

The **/integrations** endpoint allows you to create and configure new integrations.

To retrieve and update existing integrations see /integrations/{integrationId}.

Back to Integrations API Endpoint Reference.

GET

Retrieves a list of configured integrations.

Path parameters

The **/integrations** endpoint takes no parameters. It returns data for all existing integrations.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
readonly	List	Read-only details about the integration. This can include the webhook URL of the integration and authorisation details.
type_id	String	Type of integration you have created.
inputs	String	Username (**value**) and password (**key**) you have configured to authenticate with the integration.
name	String	Name of the integration you have created.
id	Integer	ID of the integration you have created.
version	String	Version of the integration you have created.
config	Object	The integration's configuration.

Examples

The following examples demonstrate typical use making a GET request to the **integrations** endpoint:

Request example

Example cURL GET request to the instance:

```
curl -X GET \
https://example.com/integrations/api/v1/integrations \
-u phil:password123 \
```

Response example

Example response returning the configured integrations:


```
[
  {
    "id": 42,
    "name": "Webhook1",
    "type_id": "Webhook",
    "version": "1.14",
    "config": {
      "single_instance_only": false,
      "category": "monitoring",
      "description": "A webhook integration to allow events to be sent via
generic REST and processed by Moogsoft AIOps",
      "display_name": "Slack",
      "type_id": "Webhook",
      "version": "1.14",
      "properties": [
        {
          "moobot_visibility": true,
          "property": "gatewayURL",
          "value": "jira"
        }
      ],
      "validations": [
        {
          "name": "jira_availability",
          "type": "HTTP",
          "method": "GET",
          "uri": "http://validator.com",
          "headers": {
            "additionalProp1": "string",
            "additionalProp2": "string",
            "additionalProp3": "string"
          },
          "params": {
            "additionalProp1": "string",
            "additionalProp2": "string",
            "additionalProp3": "string"
          },
          "body": "string",
          "errorMessage": "Provided credentials were rejected by the host"
        }
      ],
      "alert_url_tools": [
        {
          "additionalProp1": {},
          "additionalProp2": {},
          "additionalProp3": {}
        }
      ],
      "custom_fields": [
        {
          "additionalProp1": {},
          "additionalProp2": {},
          "additionalProp3": {}
        }
      ],
      "link_definitions": [
        {
          "additionalProp1": {},
          "additionalProp2": {},
```

```
    "additionalProp3": {}
  }
],
"moolets": [
  {
    "additionalProp1": {},
    "additionalProp2": {},
    "additionalProp3": {}
  }
],
"sig_url_tools": [
  {
    "additionalProp1": {},
    "additionalProp2": {},
    "additionalProp3": {}
  }
],
"sitroom_plugins": [
  {
    "additionalProp1": {},
    "additionalProp2": {},
    "additionalProp3": {}
  }
],
"config": {
  "monitor": {
    "additionalProp1": {},
    "additionalProp2": {},
    "additionalProp3": {}
  },
  "constants": {
    "additionalProp1": {
      "additionalProp1": {},
      "additionalProp2": {},
      "additionalProp3": {}
    },
    "additionalProp2": {
      "additionalProp1": {},
      "additionalProp2": {},
      "additionalProp3": {}
    },
    "additionalProp3": {
      "additionalProp1": {},
      "additionalProp2": {},
      "additionalProp3": {}
    }
  },
  "conversions": {
    "additionalProp1": {
      "input": "STRING",
      "output": "INTEGER"
    },
    "additionalProp2": {
      "input": "STRING",
      "output": "INTEGER"
    },
    "additionalProp3": {
```

```
            "input": "STRING",
            "output": "INTEGER"
          }
        },
        "filter": {
          "presend": "WebhookLam-SolutionPak.js",
          "modules": [
            "string"
          ],
          "dependencies": {
            "additionalProp1": {},
            "additionalProp2": {},
            "additionalProp3": {}
          }
        },
        "mapping": {
          "lambotOverridden": [
            "string"
          ],
          "builtInMapper": "CJsonDecoder",
          "catchAll": "overflow",
          "rules": [
            {
              "name": "signature",
              "rule": "$origin::$deviceId::$objectId",
              "conversion": "sevConverter"
            }
          ]
        }
      },
      "inputs": [
        {
          "key": "username",
          "value": "admin"
        }
      ],
      "readonly": [
        {
          "name": "url",
          "description": "URL",
          "value": "$config#proxy($config.name)"
        }
      ]
    }
  ]
```

POST

Creates an integration's configuration.

Request arguments

The POST request takes the following request payload:

Name	Type	Required	Description
type_id	String	Yes	Type of integration to add, for example **Webhook**
inputs	String	Yes	The **key** and **value** of inputs to substitute into the integration's configuration. This can include username, password, URL to poll, and timing

			intervals.
name	String	Yes	Name of the integration to add, for example **Webhook1**
version	String	Yes	Version of the integration to use. For validation purposes, this must be the most recent version.

Response

The POST request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
readonly	List	Read-only details about the integration. This can include the webhook URL of the integration and authorisation details.
type_id	String	Type of integration you have created.
inputs	String	Username (**value**) and password (**key**) you have configured to authenticate with the integration.
name	String	Name of the integration you have created.
id	Integer	ID of the integration you have created.
version	String	Version of the integration you have created.
config	Object	The integration's configuration.

Examples

The following examples demonstrate typical use making a POST request to the **integrations** endpoint:

Request example

Example cURL POST request to create a Webhook integration:

```
curl -X POST \
https://example.com/integrations/api/v1/integrations \
-u phil:password123 \
-d '{
    "type_id": "Webhook",
    "inputs": [
    {
        "name": "username",
        "value": "myusername"
    },
    {
        "name": "password",
        "value": "mypassword"
    }
```

```
        ],
        "name": "Webhook1",
        "version": "1.14"
        }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

Example response returning the new Webhook integration's configuration:

```
{
    "id": 4,
    "name": "Webhook2",
    "type_id": "Webhook",
    "version": "1.14",
    "config": {
        "category": "monitoring",
        "description": "A webhook integration to allow events to be sent via
generic REST.",
        "display_name": "Webhook",
        "type_id": "Webhook",
        "version": "1.14",
        "config": {
            "monitor": {
                "name": "Webhook Lam Monitor",
                "class": "CRestMonitor",
                "port": "$config#port()",
                "authentication_type": "basic_auth_static",
                "basic_auth_static": {
                    "username": "John.Doe",
                    "password": "Password123"
                },
                "use_ssl": false,
                "accept_all_json": true,
                "lists_contain_multiple_events": true,
                "num_threads": 5,
                "rest_response_mode": "on_receipt",
                "rpc_response_timeout": 20
            },
            "constants": {
                "severity": {
                    "CLEAR": 0,
                    "INDETERMINATE": 1,
                    "WARNING": 2,
                    "MINOR": 3,
                    "MAJOR": 4,
                    "CRITICAL": 5,
                    "0": 0,
                    "1": 1,
                    "2": 2,
                    "3": 3,
                    "4": 4,
                    "5": 5,
                    "moog_lookup_default": 1
                }
            },
            "conversions": {
                "sevConverter": {
                    "input": "STRING",
                    "output": "INTEGER",
                    "lookup": "severity"
```

```
    },
    "stringToInt": {
        "input": "STRING",
        "output": "INTEGER"
    }
},
"filter": {
    "presend": "WebhookLam-SolutionPak.js",
    "modules": [],
    "dependencies": {
        "lambot": [
            "LamBot.js",
            "WebhookLam-SolutionPak.js"
        ],
        "contrib": []
    }
},
"mapping": {
    "lambotOverridden": [],
    "catchAll": "overflow",
    "rules": [
        {
            "name": "signature",
            "rule": "$source::$type"
        },
        {
            "name": "source_id",
            "rule": "$source_id"
        },
        {
            "name": "external_id",
            "rule": "$external_id"
        },
        {
            "name": "manager",
            "rule": "$manager"
        },
        {
            "name": "source",
            "rule": "$source"
        },
        {
            "name": "class",
            "rule": "$class"
        },
        {
            "name": "agent",
            "rule": "$LamInstanceName"
        },
        {
            "name": "agent_location",
            "rule": "$agent_location"
        },
        {
            "name": "type",
            "rule": "$type"
        },
```

```
                    {
                        "name": "severity",
                        "rule": "$severity",
                        "conversion": "sevConverter"
                    },
                    {
                        "name": "description",
                        "rule": "$description"
                    },
                    {
                        "name": "agent_time",
                        "rule": "$agent_time",
                        "conversion": "stringToInt"
                    }
                ]
            }
        },
        "ha_profile": "active_active"
    },
    "inputs": [
        {
            "key": "username",
            "value": "password"
        }
    ],
    "readonly": [
        {
            "name": "url",
            "description": "URL:",
            "value": "https://example.com/integrations/api/v1/events/webhook2"
        },
        {
            "name": "userid",
            "description": "User ID:",
            "value": "myusername"
        },
        {
            "name": "readonly_password",
            "description": "Password:",
            "value": "mypassword"
        },
        {
            "name": "auth",
            "description": "Base64 Encoded Auth:",
            "value": "Basic YWRtaW46"
        }
    ]
}
```

/integrations/{integrationId}

The **`/integrations/{integrationId}`** endpoint allows you to retrieve and update existing integrations.

To create a new integration see /integrations.

Back to Integrations API Endpoint Reference.

GET

Retrieves a specific integration's configuration.

Path parameters

The GET request takes the following path parameter:

Name	Type	Required	Description
integrationId	Integer	Yes	ID of the integration to retrieve. You can obtain an integration's ID by executing a GET request to **/integrations**.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
readonly	List	Read-only details about the integration. This can include the webhook URL of the integration and authorization details.
type_id	String	Type of integration.
inputs	String	Username (**value**) and password (**key**) you have configured to authenticate with the integration.
name	String	Name of the integration.
id	Integer	ID of the integration.
version	String	Version of the integration. For validation purposes, this must be the most recent version.
config	Object	Integration's configuration.

Examples

The following examples demonstrate making a GET request to the **integrations/{integrationId}** endpoint:

Request example

Example cURL request for details of the integration with the ID **"3"**:

```
curl \
https://example.com/integrations/api/v1/integrations/3 \
-u phil:password123 \
```

Response example

A successful request returns the HTTP code 200 and no response text.

Example response returning the integration's details:

```
{
    "id": 3,
```

```
"name": "DynatraceAPMPolling1",
"type_id": "dynatrace_apm_lam",
"version": "2.3",
"config": {
    "config": {
        "filter": {
            "modules": [],
            "presend": "DynatraceApmLam.js",
            "dependencies": {
                "lambot": [
                    "LamBot.js",
                    "DynatraceApmLam.js"
                ],
                "contrib": []
            }
        },
        "mapping": {
            "rules": [
                {
                    "name": "signature",
                    "rule": "$systemprofile :: $rule"
                },
                {
                    "name": "source_id",
                    "rule": "Dynatrace APM"
                },
                {
                    "name": "external_id",
                    "rule": "$id"
                },
                {
                    "name": "manager",
                    "rule": "Dynatrace Apm"
                },
                {
                    "name": "source",
                    "rule": "$source"
                },
                {
                    "name": "class",
                    "rule": "$rule"
                },
                {
                    "name": "agent",
                    "rule": "$LamInstanceName"
                },
                {
                    "name": "agent_location",
                    "rule": "$LamInstanceName"
                },
                {
                    "name": "type",
                    "rule": "$state"
                },
                {
                    "name": "severity",
                    "rule": "$severity",
                    "conversion": "sevConverter"
                },
                {
                    "name": "description",
```

```
                "rule": "$message"
            },
            {
                "name": "agent_time",
                "rule": "$start",
                "conversion": "timeConverter"
            }
        ],
        "catchAll": "overflow",
        "lambotOverridden": [
            "custom_info.overflow",
            "source"
        ]
    },
    "monitor": {
        "name": "DynatraceApm Lam Monitor",
        "class": "CDynatraceApmMonitor",
        "targets": {
            "target1": {
                "url": "https://localhost:8021",
                "filter": {
                    "state": "InProgress",
                    "profileName": "nam",
                    "incidentRule": "rul"
                },
                "timeout": 120,
                "password": "def",
                "username": "abc",
                "disable_certificate_validation": true
            }
        },
        "max_retries": -1,
        "retry_interval": 60,
        "request_interval": 60
    },
    "constants": {
        "severity": {
            "severe": 5,
            "warning": 2,
            "informational": 1
        }
    },
    "conversions": {
        "stringToInt": {
            "input": "STRING",
            "output": "INTEGER"
        },
        "sevConverter": {
            "input": "STRING",
            "lookup": "severity",
            "output": "INTEGER"
        },
        "timeConverter": {
            "input": "STRING",
            "output": "INTEGER",
            "timeFormat": "yyyy-MM-dd'T'HH:mm:ss.SSS"
        }
```

```
                }
            },
            "moolets": [],
            "type_id": "dynatrace_apm_lam",
            "version": "2.3",
            "category": "monitoring",
            "ha_profile": "active_passive",
            "description": "An integration which enables Moogsoft AIOps to ingest
    events from Dynatrace APM.",
            "display_name": "Dynatrace APM (Polling)"
        },
        "inputs": [
            {
                "key": "targets",
                "value": [
                    {
                        "url": "https://localhost:8021",
                        "filter": {
                            "state": "InProgress",
                            "profile_name": "nam",
                            "incident_rule": "rul"
                        },
                        "password": "def",
                        "username": "abc"
                    }
                ]
            },
            {
                "key": "timing",
                "value": {
                    "timeout": 120,
                    "retry_interval": 60,
                    "request_interval": 60
                }
            }
        ],
        "readonly": null
    }
```


PUT

Updates an integration's configuration. Integrations with the ID you specify are unavailable during the update. When the update completes, they automatically resume.

Request arguments

The PUT request takes the following request arguments:

Name	Type	Required	Description
integrationId	Integer	Yes	ID of the integration to update.
type_id	String	Yes	Type of integration to add, for example **Webhook**.
inputs	String	Yes	The **key** and **value** of inputs to substitute into the integration's configuration. This can include username, password, URL to poll, and timing intervals.
name	String	Yes	Name of the integration to add, for example **Webhook1**.
version	String	Yes	Version of the integration to use. For validation purposes, this must

			be the most recent version.

Response

The PUT request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
readonly	List	Read-only details about the integration. This can include the webhook URL of the integration and authorisation details.
type_id	String	Type of integration you have updated.
inputs	String	Username (**value**) and password (**key**) you have configured to authenticate with the integration.
name	String	Name of the integration you have updated.
id	Integer	ID of the integration you have updated.
version	String	Version of the integration you have updated.
config	Object	The integration's configuration.

Examples

The following examples demonstrate making a PUT request to the **integrations/{integrationId}** endpoint:

Request example

Example cURL PUT request to update the **value** and **password** parameters for a Webhook integration. In this example, the Webhook's ID is **2**:

```
curl -X PUT \
https://example.com/integrations/api/v1/integrations/2 \
-u phil:password123 \
-d '{
      "type_id": "Webhook",
      "inputs": [
      {
          "name": "username",
          "value": "Jane.Doe"
      },
      {
          "name": "password",
          "value": "Password123"
      }
      ],
      "name": "Webhook1",
      "version": "1.14"
    }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

Example response returning the updated integration's details:


```
{
    "id": 2,
    "name": "Webhook2",
    "type_id": "Webhook",
    "version": "1.14",
    "config": {
        "category": "monitoring",
        "description": "A webhook integration to allow events to be sent via
generic REST and processed by Moogsoft AIOps.",
        "display_name": "Webhook",
        "type_id": "Webhook",
        "version": "1.14",
        "config": {
            "monitor": {
                "name": "Webhook Lam Monitor",
                "class": "CRestMonitor",
                "port": "$config#port()",
                "authentication_type": "basic_auth_static",
                "basic_auth_static": {
                    "username": "<username>",
                    "password": "<password>"
                },
                "use_ssl": false,
                "accept_all_json": true,
                "lists_contain_multiple_events": true,
                "num_threads": 5,
                "rest_response_mode": "on_receipt",
                "rpc_response_timeout": 20
            },
            "constants": {
                "severity": {
                    "CLEAR": 0,
                    "INDETERMINATE": 1,
                    "WARNING": 2,
                    "MINOR": 3,
                    "MAJOR": 4,
                    "CRITICAL": 5,
                    "0": 0,
                    "1": 1,
                    "2": 2,
                    "3": 3,
                    "4": 4,
                    "5": 5,
                    "moog_lookup_default": 1
                }
            },
            "conversions": {
                "sevConverter": {
                    "input": "STRING",
                    "output": "INTEGER",
                    "lookup": "severity"
                },
                "stringToInt": {
                    "input": "STRING",
                    "output": "INTEGER"
                }
```

```
},
"filter": {
    "presend": "WebhookLam-SolutionPak.js",
    "modules": [],
    "dependencies": {
        "lambot": [
            "LamBot.js",
            "WebhookLam-SolutionPak.js"
        ],
        "contrib": []
    }
},
"mapping": {
    "lambotOverridden": [],
    "catchAll": "overflow",
    "rules": [
        {
            "name": "signature",
            "rule": "$source::$type"
        },
        {
            "name": "source_id",
            "rule": "$source_id"
        },
        {
            "name": "external_id",
            "rule": "$external_id"
        },
        {
            "name": "manager",
            "rule": "$manager"
        },
        {
            "name": "source",
            "rule": "$source"
        },
        {
            "name": "class",
            "rule": "$class"
        },
        {
            "name": "agent",
            "rule": "$LamInstanceName"
        },
        {
            "name": "agent_location",
            "rule": "$agent_location"
        },
        {
            "name": "type",
            "rule": "$type"
        },
        {
            "name": "severity",
            "rule": "$severity",
            "conversion": "sevConverter"
        },
```

```
                    {
                        "name": "description",
                        "rule": "$description"
                    },
                    {
                        "name": "agent_time",
                        "rule": "$agent_time",
                        "conversion": "stringToInt"
                    }
                ]
            }
        },
        "ha_profile": "active_active"
    },
    "inputs": [
        {
            "key": "Jane.Doe",
            "value": "MyPassword"
        }
    ],
    "readonly": [
        {
            "name": "url",
            "description": "URL:",
            "value": "https://example.com/integrations/api/v1/events/webhook2"
        },
        {
            "name": "userid",
            "description": "User ID:",
            "value": "Username"
        },
        {
            "name": "readonly_password",
            "description": "Password:",
            "value": "Password123"
        },
        {
            "name": "auth",
            "description": "Base64 Encoded Auth:",
            "value": "Basic YWRtaW46"
        }
    ]
}
```

/integrations/{integrationId}/status

The **`/integrations/{integrationID}/status`** endpoint allows you to check and update the status of an integration.

Back to Integrations API Endpoint Reference.

GET

Retrieves the status of an integration.

Path parameters

The GET request takes the following path parameter:

Name	Type	Required	Description
`integrationId`	Integer	Yes	ID of the integration to retrieve. You can obtain an integration's ID

			by executing a GET request to **`/integrations`**.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`instances`	String	URL of the instances in an object of Broker IDs that map to the integration status info.
`global`	Object	Contains **`integration_config_id`** and **`status`**.
`integration_config_id`	Integer	ID of the integration.
`status`	String	Current status of the integration.

Examples

The following examples demonstrate making a GET request to the endpoint **`/integrations/{integrationId}/status`**:

Request example

Example cURL GET request to retrieve the current status of the integration with the ID **`"6"`**:

```
curl \
https://example.com/integrations/api/v1/integrations/6/status \
-u phil:password123 \
-H "accept: application/json"
```

Response example

A successful request returns the HTTP code 200 and no response text.

Example response returning the integration's current status:

```
{
    "global":
    {
        "integration_config_id": 6,
        "status": "running"
    },
    "instances":
    {
        "Broker_3a52b7ef_8bad_41cc_8b36_cbb9c2aa3a9e":
        {
            "integration_name": "Azure1",
            "status": "running",
            "last_heartbeat": 1567789762645
        }
```

```
        }
    }
```

PUT

Updates the status of an integration. You can use this to start and stop integrations. An integration can run on multiple brokers; when assigning an integration to a broker, Cisco Crosswork Situation Manager favours the broker(s) running the least integrations.

Request arguments

The PUT request takes the following request argument:

Name	Type	Required	Description
`integrationId`	Integer	Yes	ID of the integration to update.

Response

The PUT request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Required	Description
`command`	String	Yes	Status command you want to make. Choose from **`start`** and **`stop`**.
`target`	String	Yes, unless using the **`start`** command	Broker on which to run or stop the integration. If unspecified, Cisco Crosswork Situation Manager starts the integration on the broker running the least integrations.

Examples

The following examples demonstrate making a PUT request to the endpoint **`/integrations/{integrationId}/status`**:

Request example

Example cURL PUT request to update an integration.

```
curl -X PUT \
https://example.com/integrations/api/v1/integrations/6/status \
-u phil:password123 \
-H "Content-Type: application/json; charset=UTF-8" \
-d '{ "command": "start" }'
```

Response example

A successful request returns the HTTP code 200 and no response text.

Export and Import Integrations

You can use the integrations API to migrate integration configurations across your Cisco Crosswork Situation Manager instances, allowing you to swiftly set up new integrations while keeping previous configurations intact.

This example workflow combines queries to multiple endpoints to create an export/import workflow, exporting an integration from one instance and then importing it into another instance.

Before you begin

Before you start the workflow, ensure you have met the following requirements:

- You have access to two separate instances of Cisco Crosswork Situation Manager and an integration on one of these.
- You have running brokers on the destination instance.
- You have the ID of the integration(s) you want to export from your first instance. The ID displays in the URL when you open the integration in your browser. For example, https:/example.com/#/integrations/integration-details/13 has the ID 13.

Export the integration

The first step is to export the integration's details. Using basic authentication, make a GET request to your first instance's /integrations/{integrationId} endpoint to retrieve its payload:

```
curl \
https://instance1.com/integrations/api/v1/integrations/3 \
-u phil:password123
```

The payload in the response returns the integration's details. For example:

```
{
    "id": 3,
    "name": "DynatraceAPMPolling1",
    "type_id": "dynatrace_apm_lam",
    "version": "2.3",
    "config": {
        "config": {
            "filter": {
                "modules": [],
                "presend": "DynatraceApmLam.js",
                "dependencies": {
                    "lambot": [
                        "LamBot.js",
                        "DynatraceApmLam.js"
                    ],
                    "contrib": []
                }
            },
            "mapping": {
                "rules": [
                    {
                        "name": "signature",
                        "rule": "$systemprofile :: $rule"
                    },
                    {
                        "name": "source_id",
                        "rule": "Dynatrace APM"
                    },
                    {
                        "name": "external_id",
```

```
                "rule": "$id"
            },
            {
                "name": "manager",
                "rule": "Dynatrace Apm"
            },
            {
                "name": "source",
                "rule": "$source"
            },
            {
                "name": "class",
                "rule": "$rule"
            },
            {
                "name": "agent",
                "rule": "$LamInstanceName"
            },
            {
                "name": "agent_location",
                "rule": "$LamInstanceName"
            },
            {
                "name": "type",
                "rule": "$state"
            },
            {
                "name": "severity",
                "rule": "$severity",
                "conversion": "sevConverter"
            },
            {
                "name": "description",
                "rule": "$message"
            },
            {
                "name": "agent_time",
                "rule": "$start",
                "conversion": "timeConverter"
            }
        ],
        "catchAll": "overflow",
        "lambotOverridden": [
            "custom_info.overflow",
            "source"
        ]
    },
    "monitor": {
        "name": "DynatraceApm Lam Monitor",
        "class": "CDynatraceApmMonitor",
        "targets": {
            "target1": {
                "url": "https://localhost:8021",
                "filter": {
                    "state": "InProgress",
                    "profileName": "nam",
                    "incidentRule": "rul"
                },
                "timeout": 120,
                "password": "def",
                "username": "abc",
```

```
                        "disable_certificate_validation": true
                    }
                },
                "max_retries": -1,
                "retry_interval": 60,
                "request_interval": 60
            },
            "constants": {
                "severity": {
                    "severe": 5,
                    "warning": 2,
                    "informational": 1
                }
            },
            "conversions": {
                "stringToInt": {
                    "input": "STRING",
                    "output": "INTEGER"
                },
                "sevConverter": {
                    "input": "STRING",
                    "lookup": "severity",
                    "output": "INTEGER"
                },
                "timeConverter": {
                    "input": "STRING",
                    "output": "INTEGER",
                    "timeFormat": "yyyy-MM-dd'T'HH:mm:ss.SSS"
                }
            }
        },
        "moolets": [],
        "type_id": "dynatrace_apm_lam",
        "version": "2.3",
        "category": "monitoring",
        "ha_profile": "active_passive",
        "description": "An integration which enables Moogsoft AIOps to ingest
events from Dynatrace APM.",
        "display_name": "Dynatrace APM (Polling)"
    },
    "inputs": [
        {
            "key": "targets",
            "value": [
                {
                    "url": "https://localhost:8021",
                    "filter": {
                        "state": "InProgress",
                        "profile_name": "nam",
                        "incident_rule": "rul"
                    },
                    "password": "def",
                    "username": "abc"
                }
            ]
        },
        {
```

```
            "key": "timing",
            "value": {
                "timeout": 120,
                "retry_interval": 60,
                "request_interval": 60
            }
        }
    ],
    "readonly": null
}
```

Import the integration

Now that you have the integration's details from the payload you can import them into your second instance and create a new integration.

Make a POST request to your destination instance's /integrations endpoint, using the whole payload from the GET request. You do not need to omit the **id** parameter as the POST request ignores it. For example:

```
curl -X POST \
https://instance2.com/integrations/api/v1/integrations \
-H 'Content-Type: application/json' \
-u phil:password123 \
-d '{
    "id": 3,
    "type_id": "dynatrace_apm_lam",
    "inputs": [
        {
            "key": "targets",
            "value": [
                {
                    "url": "https://localhost:8021",
                    "filter": {
                        "state": "InProgress",
                        "profile_name": "nam",
                        "incident_rule": "rul"
                    },
                    "password": "def",
                    "username": "abc"
                }
            ]
        },
        {
            "key": "timing",
            "value": {
                "timeout": 120,
                "retry_interval": 60,
                "request_interval": 60
            }
        }
    ],
    "name": "DynatraceAPMPolling1",
    "version": "2.3"
    }'
```

A successful response returns a payload containing the new integration's details.

Start the integration

Having exported the integration, the final step is to start it up.

Make a PUT request to your destination instance's /integrations/{integrationId}/status endpoint:

```
curl -X PUT \
https://instance2.com/integrations/api/v1/integrations/6/status \
-H 'Content-Type: application/json' \
-u phil:password123 \
-d '{
      "command": "start"
    }'
```

The integration is now ready to use on your second instance.

Manage Integration States

The Integrations Controller stores the states of UI integrations in **<moog_intdb>.integration_state**. To modify the state of an integration, you can either run it or use the Integrations API. If you use the Integrations API, you can modify the data ingestion information for a process group.

This topic provides an example state management workflow in which you combine queries to check an integration's state, then set its last poll time back an hour.

Before you begin

Before you start the workflow, ensure you have met the following requirements:

- You have the name of the integration you want to modify.
- The integration you want to modify is either running or pre-populated in a non-running state.

Check the integration state

Using basic authentication, make a GET request to retrieve the status of the integration, which contains the last poll time in milliseconds. In this example, **ZabbixPolling2** is the name of the integration:

```
curl -u John.Doe:MyPassword https://<host>/integrations/api/v1/integration-
states/ZabbixPolling2
```

The response indicates the last poll time. For example:

```
{
    "target1": {
        "last_poll_time_ms": 1574758921000
    }
}
```

Note

If you are checking the state of a LAM, in place of the integration name you use the HA group name. You can find this in the LAM configuration file under the **ha** section. See LAMS and High Availability for more information.

Modify the integration state

Now that you have the last poll time you can change the timestamp. Make a PUT request to the same endpoint with the new value for **last_poll_time_ms**:

```
curl -X PUT \
 https://<host>/integrations/api/v1/integration-states/ZabbixPolling2_1 \
```

```
 -u John.Doe:MyPassword
  -H 'Content-Type: application/json' \
  -d '{
    "target1": {
        "last_poll_time_ms": 1574762521000
    }
}'
```

For a running integration, upon successful completion your changes instantly apply.

If the integration is not running or does not yet exist, the state is applied once the integration starts. See /integrations/{integrationId}/status for more information.

Topologies API

The Topologies API allows you to create, modify, retrieve and delete topologies and their nodes and links. You can use the clone and replace endpoints to update a copy of an existing topology and then replace an active topology with the updated version.

You can use the Topologies API to create and manage small topologies, but this is impractical for large topologies. If your topology .csv file is larger than 40 MB Cisco recommends using the Topology Loader utility.

See Load a Topology for information on the loader utility. Contact Cisco Support if you experience difficulties or need further guidance.

Endpoints

See Topologies API Endpoint Reference for details of all the Topologies API endpoints.

API definition

All Topologies API requests use the following URL format, where **<server>** is the hostname of the machine running the Cisco Crosswork Situation Manager UI:

`https://<server>/api/v1/topologies/<endpoint>`

Examples:

`https://example.com/api/v1/topologies`

`https://example.com/api/v1/topologies/{topologyName}/nodes`

`https://example.com/api/v1/topologies/{topologyName}/links/{sourceNode}`

API behavior

The following behavior and restrictions apply to the Topologies API:

- The endpoints do not allow for filtering, sorting, limiting or pagination.
- To change the name of a topology, use the /topologies/{topologyName}/replace endpoint.
- You cannot change the name of a node. This is by design, to preserve data consistency.
- You cannot replace, rename or delete a topology or set it to "inactive" if it's being used to filter a Recipe.
- All POST and PUT requests are batched automatically by the API.
- Topology names and node names are saved in lowercase.

- GET and DELETE operations are case insensitive.
- If you attempt to create, retrieve, modify or delete nodes or links for a topology that does not exist, you will receive a 400 "bad request" response.

For more information on the behavior of individual endpoints, see the Topologies API Endpoint Reference.

Authentication

To use the Topologies API, you must have the super_privileges permission. See Role Permissions for more information.

All requests require a basic authentication header.

Topologies API Endpoint Reference

This is a reference list for the Topologies API endpoints. Follow the links to see the details of each endpoint.

All of the endpoints use basic authorization.

Topologies

The following endpoints relate to topologies:

- /topologies: Creates and updates multiple topologies, and retrieves all active topologies.
- /topologies/inactive: Retrieves all inactive topologies.
- /topologies/{topologyName}: Retrieves and deletes a single topology.
- /topologies/{topologyName}/clone: Clones a topology.
- /topologies/{topologyName}/replace: Replaces an existing topology with another topology, or renames a topology.

Nodes

The following endpoints relate to topology nodes:

- /topologies/{topologyName}/nodes: Creates, retrieves, updates, and deletes multiple nodes.
- /topologies/{topologyName}/nodes/{nodeName}: Retrieves a single node.

Links

The following endpoints relate to topology links:

- /topologies/{topologyName}/links: Creates, retrieves, and deletes multiple links.
- /topologies/{topologyName}/links/{nodeName}: Retrieves and deletes all links for a node.
- /topologies/{topologyName}/links/{sourceNode}/{sinkNode}: Retrieves a link between two nodes.

Situations

The following endpoints relate to topologies and Situations:

- /situation/{situationID}/topologies: Retrieves the topologies related to the alerts in a specified Situation.
- getSituationTopology: Retrieves the node and link details for a specified Situation and topology.

/topologies

The **`/topologies`** endpoint allows you to create, retrieve and update one or more topologies.

To retrieve and delete a single existing topology, see /topologies/{topologyName}.

Back to Topologies API Endpoint Reference.

GET

Retrieves details of all active topologies.

Path parameters

The GET request takes no parameters. It returns data for all active topologies in the system.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`name`	String	Name of the topology.
`description`	String	Description of the topology.
`active`	Boolean	Whether the topology is active (**`true`**) or inactive (**`false`**).

Examples

The following examples demonstrate making a GET request to the **`/topologies`** endpoint.

Request example

Example cURL GET request to return details for all active topologies:

```
curl -X GET 'https://example.com/api/v1/topologies' -u phil:password123
```

Response example

Example response returning the active topology details:

```
[
    {
        "name": "host",
        "description": "Host-based topology",
        "active": true
    },
    {
        "name": "location",
        "description": "Location-based topology",
        "active": true
```

```
    }
]
```

POST

Creates one or more topologies.

Request arguments

The POST request takes the following request payload:

Name	Type	Required	Description
name	String	Yes	Name of the topology. Must be less than 256 characters.
description	String	No	Description of the topology. Must be less than 1001 characters.
active	Boolean	No	Whether the topology is active (**true**) or inactive (**false**). Default is **false**.

Response

The POST request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	Name of the topology.
description	String	Description of the topology.
active	Boolean	Whether the topology is active (**true**) or inactive (**false**).

If you send an existing name in the request, it will be ignored and returned in the response.

Examples

The following examples demonstrate making a POST request to the **/topologies** endpoint.

Request example

Example cURL POST request to create two topologies named "host" and "location":

```
curl -X POST 'https://example.com/api/v1/topologies' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u phil:password123 \
-d '[{"name":"host","description":"Host-based
topology","active":true},{"name":"location","description":"Location-based
topology","active":true}]'
```

Response example

A successful request returns the HTTP code 200 and no response text.

Example response returning the new topologies:

```
[
    {
        "name": "host",
        "description": "Host-based topology",
        "active": true
    },
    {
        "name": "location",
        "description": "Location-based topology",
        "active": true
    }
]
```

PUT

Updates one or more topologies.

Request arguments

The PUT request takes the following request payload:

Name	Type	Required	Description
name	String	Yes	Name of the topology. Must be less than 256 characters. You cannot update the topology name. To rename a topology use the /topologies/{topologyName}/replace endpoint.
description	String	No	Description of the topology. Must be less than 1001 characters.
active	Boolean	No	Whether the topology is active (**true**) or inactive (**false**). Default is **false**. You cannot set a topology to inactive if it's being used to filter a Recipe.

Response

The PUT request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	Name of the topology.
description	String	Description of the topology.
active	Boolean	Whether the topology is active (**true**) or inactive (**false**).

Examples

The following examples demonstrate making a PUT request to the **/topologies** endpoint.

Request example

Example cURL PUT request to update the descriptions of two topologies named "host" and "location":

```
curl -X PUT 'https://example.com/api/v1/topologies' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u phil:password123 \
-d '[{"name":"host","description":"Host-based network
topology","active":false},{"name":"location","description":"Location-based
network topology","active":false}]'
```

Response example

A successful request returns the HTTP code 200 and no response text.

Example response returning the updated topologies:

```
[
    {
        "name": "host",
        "description": "Host-based network topology",
        "active": false
    },
    {
        "name": "location",
        "description": "Location-based network topology",
        "active": false
    }
]
```

/topologies/inactive

The **`/topologies/inactive`** endpoint allows you to retrieve all inactive topologies.

To retrieve all active topologies see /topologies.

Back to Topologies API Endpoint Reference.

GET

Retrieves details of all inactive topologies.

Path parameters

The GET request takes no parameters. It returns data for all inactive topologies in the system.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
`name`	String	Name of the topology.
`description`	String	Description of the topology.

active	Boolean	Whether the topology is active (**true**) or inactive (**false**).

Examples

The following examples demonstrate making a GET request to the **/topologies/inactive** endpoint.

Request example

Example cURL GET request to return details for all inactive topologies:

```
curl -X GET 'https://example.com/api/v1/topologies/inactive' -u phil:password123
```

Response example

Example response returning the inactive topology details:

```
[
    {
        "name": "host",
        "description": "Host-based topology",
        "active": false
    },
    {
        "name": "location",
        "description": "Location-based topology",
        "active": false
    }
]
```

/topologies/{topologyName}

The **/topologies/{topologyName}** endpoint allows you to retrieve and delete a single topology.

To create, retrieve and update multiple topologies see /topologies.

Back to Topologies API Endpoint Reference.

GET

Retrieves details of a specified topology.

Path parameters

The GET request takes the following path parameter:

Name	Type	Required	Description
topologyName	String	Yes	Name of the topology. You can obtain all topology names by executing a GET request to **/topologies**.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	Name of the topology.

description	String	Description of the topology.
active	Boolean	Whether the topology is active (**true**) or inactive (**false**).

Example

The following example demonstrates making a GET request to the **/topologies/{topologyName}** endpoint.

Request example

Example cURL request for details of the topology with name "host":

```
curl -X GET \
https://example.com/api/v1/topologies/host \
-u phil:password123 \
```

Response example

Example response returning the topology's details:

```
{
    "name": "host",
    "description": "Host-based topology",
    "active": true
}
```

DELETE

Deletes a specified topology. Note the following:

- You cannot delete a topology if it's being used to filter a Recipe.
- Deleting a topology will impact your ability to restore associated Recipes in future. In order to restore a Recipe that filters on a named topology, the topology must still exist in Cisco Crosswork Situation Manager.

Request arguments

The DELETE request takes no arguments. It deletes the specified topology and any associated nodes and links.

Response

The DELETE request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Example

The following example demonstrates making a DELETE request to the **/topologies/{topologyName}** endpoint.

Request example

Example cURL request to delete a topology with name "host":

```
curl -X DELETE \
https://example.com/api/v1/topologies/host \
-u phil:password123
```

Response example

A successful request returns the HTTP code 204 and no content.

/topologies/{topologyName}/clone

The **`/topologies/{topologyName}/clone`** endpoint allows you to clone a topology. You can use the clone and replace topologies endpoints to update a copy of an existing topology and then replace a topology with the updated version.

To replace an existing topology with a cloned topology, see /topologies/{topologyName}/replace.

Back to Topologies API Endpoint Reference.

POST

Clones a topology.

Request arguments

The POST request takes the following request payload:

Name	Type	Required	Description
`name`	String	Yes	Name for the cloned topology.

Response

The POST request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return a JSON object containing the following:

Name	Type	Description
`name`	String	Name of the topology.
`description`	String	Description of the topology.
`active`	Boolean	Whether the topology is active (**`true`**) or inactive (**`false`**). Topology clones are set to inactive. To change the active status send a PUT request to the /topologies endpoint.

Example

The following example demonstrates making a POST request to the **`/topologies/{topologyName}/clone`** endpoint.

Request example

Example cURL POST request to clone the "host" topology and name the clone "host_new":

```
curl -X POST 'https://example.com/api/v1/topologies/host/clone' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u phil:password123 \
-d '{"name":"host_new"}'
```

Response example

Example response returning the cloned topology:

```
[
    {
        "name": "host_new",
        "description: "Host-based topology",
        "active: false
    }
]
```

/topologies/{topologyName}/replace

The **`/topologies/{topologyName}/replace`** endpoint allows you to replace an existing topology with another topology. This process deletes the original topology. You can use the clone and replace topologies endpoints to update a copy of an existing topology and then replace a topology with the updated version.

You can also use this endpoint to rename a topology.

Back to Topologies API Endpoint Reference.

PUT

Replaces an existing topology with another topology, or renames a topology. Provide the **`{topologyName}`** in the endpoint according to the desired function:

- Replace: The name of the existing topology to replace.
- Rename: The new topology name.

When a topology is replaced:

- The original topology and its nodes and links are deleted.
- Alerts that reference the original topology are updated to reference the replacement topology.
- If the replacement topology is active, its processing state in the database is set to outdated. This triggers the graph analyser process to run as part of the Housekeeper Moolet. See Topologies.

Request arguments

The PUT request takes the following request payload:

Name	Type	Required	Description
`name`	String	Yes	Replace: Name of the replacing topology. Rename: The topology to rename.
`active`	Boolean	No	Sets the replaced or renamed topology to active (**`true`**) or inactive (**`false`**). Replaced topologies take the active status of the replacing topology by default.

Response

The PUT request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

The request fails if any of the following are true:

- **name** is being used to filter a Recipe, or does not exist.
- **{topologyName}** is being used to filter a Recipe and you are trying to make it inactive.
- Successful requests return a JSON object containing the following:

Name	Type	Description
name	String	Name of the topology.
description	String	Description of the topology.
active	Boolean	Whether the topology is active (**true**) or inactive (**false**).

Example

The following example demonstrates making a PUT request to the **/topologies/{topologyName}/replace** endpoint.

Request example

Example cURL PUT request to replace the "host" topology with the "host_new" topology and set its status to active:

```
curl -X PUT 'https://example.com/api/v1/topologies/host/replace' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u phil:password123 \
-d '{"name":"host_new","active":true}'
```

In this example, if there is no topology named "host" the "host_new" topology is renamed "host".

Response example

Example response returning the newly replaced topology:

```
[
    {
        "name": "host_new",
        "description": "Host-based topology",
        "active": true
    }
]
```

/topologies/{topologyName}/nodes

The **/topologies/{topologyName}/nodes** endpoint allows you to create, retrieve, update and delete one or more topology nodes.

To retrieve a single node for a topology, see /topologies/{topologyName}/nodes/{nodeName}.

Back to Topologies API Endpoint Reference.

GET

Retrieves details of all nodes for a specified topology.

Path parameters

The GET request takes the following path parameter:

Name	Type	Required	Description
topologyName	String	Yes	Name of the topology. You can obtain all topology names by executing a GET request to **/topologies**.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	Name of the node.
description	String	Description of the node.

Example

The following example demonstrates making a GET request to the **/topologies/{topologyName}/nodes** endpoint.

Request example

Example cURL GET request to return node details for the "host" topology:

```
curl -X GET 'https://example.com/api/v1/topologies/host/nodes' -u
phil:password123
```

Response example

Example response returning the topology's node details:

```
[
    {
        "name": "node1",
        "description": "First node"
    },
    {
        "name": "node2",
        "description": "Second node"
    }
]
```

POST

Creates one or more nodes in a topology.

Request arguments

The POST request takes the following request payload:

Name	Type	Required	Description

name	String	Yes	Name of the node.
description	String	No	Description of the node.

Response

The POST request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	Name of the node.
description	String	Description of the node.

If you send an existing name in the request, it will be ignored and returned in the response.

Example

The following example demonstrates making a POST request to the **/topologies/{topologyName}/nodes** endpoint.

Request example

Example cURL POST request to create two nodes in the "host" topology:

```
curl -X POST 'https://example.com/api/v1/topologies/host/nodes' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u phil:password123 \
-d '[{"name":"node1","description":"First
node"},{"name":"node2","description":"Second node"}]'
```

Response example

Example response returning the new topologies:

```
[
    {
        "name": "node1",
        "description: "First node"
    },
    {
        "name": "node2",
        "description: "Second node"
    }
]
```

PUT

Updates one or more nodes in a topology. You can only update node descriptions, not node names.

Request arguments

The PUT request takes the following request payload:

Name	Type	Required	Description
name	String	Yes	Name of the topology.

description	String	No	Description of the topology.

Response

The PUT request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
name	String	Name of the node.
description	String	Description of the node.

Example

The following example demonstrates making a PUT request to the **/topologies/{topologyName}/nodes** endpoint.

Request example

Example cURL PUT request to update two nodes in the "host" topology named "node1" and "node2":

```
curl -X PUT 'https://example.com/api/v1/topologies/host/nodes' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u phil:password123 \
-d '[{"name":"node1","description":"Primary
node"},{"name":"node2","description":"Secondary node"}]'
```

Response example

Example response returning the new nodes:

```
[
    {
        "name": "node1",
        "description: "Primary node"
    },
    {
        "name": "node2",
        "description: "Secondary node"
    }
]
```

DELETE

Deletes one or more nodes in a topology.

Request arguments

The DELETE request takes the following request payload:

Name	Type	Required	Description
names	Array of	Yes	One or more node names in the format

	Strings		**["nodename1","nodename2"]**.

Response

The DELETE request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Example

The following example demonstrates making a DELETE request to the **/topologies/{topologyName}/nodes** endpoint.

Request example

Example cURL request to delete the "node1" and "node2" nodes in the "host" topology:

```
curl -X DELETE 'https://example.com/api/v1/topologies/host/nodes' \
-H 'Content-Type: application/json' \
-d ["node1","node2"] \
-u phil:password123
```

Response example

A successful request returns the HTTP code 204 and no content.

/topologies/{topologyName}/nodes/{nodeName}

The **/topologies/{topologyName}/nodes/{nodeName}** endpoint allows you to retrieve a single existing node in a topology.

To create, retrieve, update and delete multiple topology nodes see /topologies/{topologyName}/nodes.

Back to Topologies API Endpoint Reference.

GET

Retrieves details of a specified topology node.

Path parameters

The GET request takes the following path parameters:

Name	Type	Required	Description
topologyName	String	Yes	Name of the topology.
nodeName	String	Yes	Name of the topology node.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description

name	String	Name of the node.
description	String	Description of the node.

If you send a nonexistent node name in the request, the HTTP code 204 is returned with no content.

Example

The following example demonstrates making a GET request to the **/topologies/{topologyName}/nodes/{nodeName}** endpoint.

Request example

Example cURL request for details of the "node1" node in the "host" topology:

```
curl -X GET \
https://example.com/api/v1/topologies/host/nodes/node1 \
-u phil:password123
```

Response example

Example response returning the topology's details:

```
{
    "name": "node1",
    "description": "Primary node"
}
```

/topologies/{topologyName}/links

The **/topologies/{topologyName}/links** endpoint allows you to create, retrieve and delete one or more links in a topology. You cannot update links, you must delete and re-add them.

To retrieve and delete all links for a node see /topologies/{topologyName}/links/{nodeName}.

Back to Topologies API Endpoint Reference.

GET

Retrieves details of all links for a specified topology.

Path parameters

The GET request takes the following path parameter:

Name	Type	Required	Description
topologyName	String	Yes	Name of the topology. You can obtain all topology names by executing a GET request to **/topologies**.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
description	String	Description of the link between the nodes.
sourceNode	String	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.

Example

The following example demonstrates making a GET request to the **/topologies/{topologyName}/links** endpoint.

Request example

Example cURL GET request to return link details for the "host" topology:

```
curl -X GET 'https://example.com/api/v1/topologies/host/links' -u
phil:password123
```

Response example

Example response returning the topology's link details:

```
[
    {
        "description: "link1",
        "sourceNode": "node1",
        "sinkNode: "node2"
    },
    {
        "description: "link2",
        "sourceNode": "node2",
        "sinkNode: "node3"
    }
]
```

POST

Creates one or more links in a topology. Creates the specified source nodes and sink nodes if they do not already exist.

Request arguments

The POST request takes the following request payload:

Name	Type	Required	Description
sourceNode	String	Yes	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Yes	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
description	String	No	Description of the link between the nodes.

Response

The POST request returns the following response:

Type	Description

HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
sourceNode	String	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
description	String	Description of the link between the nodes.

If the supplied link already exists, the HTTP code 200 is returned.

Example

The following example demonstrates making a POST request to the **/topologies/{topologyName}/links** endpoint.

Request example

Example cURL POST request to create two links in the "host" topology:

```
curl -X POST 'https://example.com/api/v1/topologies/host/links' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u phil:password123 \
-d
'[{"sourceNode":"node1","sinkNode":"node2","description":"link1"},{"sourceNode":
"node2","sinkNode":"node3","description":"link2"}]'
```

Response example

Example response returning the new links:

```
[
    {
        "description: "link1",
        "sourceNode": "node1",
        "sinkNode:" "node2"
    },
    {
        "description: "link2",
        "sourceNode": "node2",
        "sinkNode:" "node3"
    }
]
```

DELETE

Deletes one or more links in a topology.

Request arguments

The DELETE request takes the following request payload:

Name	Type	Required	Description

names	Array	Yes	One or more links in the format: **[** **{** **"sourceNode" : "node1",** **"sinkNode" : "node2"** **},** **{** **"sourceNode" : "node2",** **"sinkNode" : "node3"** **}** **]**

Response

The DELETE request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

If the request contains some existing and some non-existing links, the existing links will be deleted and non-existing links will be returned in the response.

Example

The following example demonstrates making a DELETE request to the **`/topologies/{topologyName}/links`** endpoint.

Request example

Example cURL request to delete the "link1" and "link2" links in the "host" topology:

```
curl -X DELETE 'https://example.com/api/v1/topologies/host/links' \
-H 'Content-Type: application/json' \
-d ["link1","link2"] \
-u phil:password123
```

Response example

A successful request returns the HTTP code 204 and no content.

An example response where link1 and link2 do not exist:

```
{
"message": "Some of the links could not be deleted as they did not
exist","invalidLinks":
    [
        {
            "sourceNode": "node1",
            "sinkNode": "node2"
        }
    ]
}
```

/topologies/{topologyName}/links/{nodeName}

The **`/topologies/{topologyName}/links/{nodeName}`** endpoint allows you to retrieve and delete all links for a topology node. Topology links in Cisco Crosswork Situation Manager are bidirectional so the node name can be either a source node or a sink node.

To create, retrieve and delete one or more links in a topology see /topologies/{topologyName}/links.

Back to Topologies API Endpoint Reference.

GET

Retrieves link details for the specified topology node.

Path parameters

The GET request takes the following path parameters:

Name	Type	Required	Description
topologyName	String	Yes	Name of the topology.
nodeName	String	Yes	Name of the node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
description	String	Description of the link between the nodes.
sourceNode	String	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.

Example

The following example demonstrates making a GET request to the **`/topologies/{topologyName}/links/{nodeName}`** endpoint.

Request example

Example cURL request for details of the `"node2"` links in the `"host"` topology:

```
curl -X GET \
https://example.com/api/v1/topologies/host/links/node2 \
-u phil:password123 \
```

Response example

Example response returning the links for `"node2"`:

```
[
    {
        "description": "link1",
        "sourceNode": "node2",
        "sinkNode": "node1"
```

```
        },
        {
            "description": "link2",
            "sourceNode": "node2",
            "sinkNode": "node3"
        }
    ]
```

DELETE

Deletes all links for the specified topology node.

Path parameters

The DELETE request takes no arguments. It deletes all links for the specified topology node.

Response

The DELETE request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Example

The following example demonstrates making a DELETE request to the **`/topologies/{topologyName}/links/{nodeName}`** endpoint.

Request example

Example cURL request to delete all links for the `"node1"` node on the `"host"` topology:

```
curl -X DELETE \
https://example.com/api/v1/topologies/host/links/node1 \
-u phil:password123
```

Response example

A successful request returns the HTTP code 204 and no content.

/topologies/{topologyName}/links/{sourceNode}/{sinkNode}

The **`/topologies/{topologyName}/links/{sourceNode}/{sinkNode}`** endpoint allows you to retrieve a link between two topology nodes.

To create, retrieve and delete one or more links in a topology see /topologies/{topologyName}/links.

To retrieve and delete all links for a topology node see /topologies/{topologyName}/links/{nodeName}.

Back to Topologies API Endpoint Reference.

GET

Retrieves link details for two specified topology nodes.

Path parameters

The GET request takes the following path parameters:

Name	Type	Required	Description

topologyName	String	Yes	Name of the topology.
sourceNode	String	Yes	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Yes	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.

Response

The GET request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

Successful requests return an array of JSON objects containing the following:

Name	Type	Description
description	String	Description of the link between the nodes.
sourceNode	String	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.

Example

The following example demonstrates making a GET request to the **`/topologies/{topologyName}/links/{sourceNode}/{sinkNode}`** endpoint.

Request example

Example cURL request for details of the link between `"node1"` and `"node2"` in the `"host"` topology:

```
curl -X GET \
https://example.com/api/v1/topologies/host/links/node1/node2 \
-u phil:password123
```

Response example

Example response returning the link details:

```
[
    {
        "description": "link1",
        "sourceNode": "node1",
        "sinkNode": "node2"
    }
]
```

/topologies/{topologyName}/replace

The **`/topologies/{topologyName}/replace`** endpoint allows you to replace an existing topology with another topology. This process deletes the original topology. You can use the clone and replace

topologies endpoints to update a copy of an existing topology and then replace a topology with the updated version.

You can also use this endpoint to rename a topology.

Back to Topologies API Endpoint Reference.

PUT

Replaces an existing topology with another topology, or renames a topology. Provide the **{topologyName}** in the endpoint according to the desired function:

- Replace: The name of the existing topology to replace.
- Rename: The new topology name.

When a topology is replaced:

- The original topology and its nodes and links are deleted.
- Alerts that reference the original topology are updated to reference the replacement topology.
- If the replacement topology is active, its processing state in the database is set to outdated. This triggers the graph analyser process to run as part of the Housekeeper Moolet. See Topologies.

Request arguments

The PUT request takes the following request payload:

Name	Type	Required	Description
name	String	Yes	Replace: Name of the replacing topology. Rename: The topology to rename.
active	Boolean	No	Sets the replaced or renamed topology to active (**true**) or inactive (**false**). Replaced topologies take the active status of the replacing topology by default.

Response

The PUT request returns the following response:

Type	Description
HTTP Code	HTTP status or error code indicating request success or failure. See HTTP status code definitions for more information.

The request fails if any of the following are true:

- **name** is being used to filter a Recipe, or does not exist.
- **{topologyName}** is being used to filter a Recipe and you are trying to make it inactive.

Successful requests return a JSON object containing the following:

Name	Type	Description
name	String	Name of the topology.
description	String	Description of the topology.
active	Boolean	Whether the topology is active (**true**) or inactive (**false**).

Example

The following example demonstrates making a PUT request to the **`/topologies/{topologyName}/replace`** endpoint.

Request example

Example cURL PUT request to replace the "host" topology with the "host_new" topology and set its status to active:

```
curl -X PUT 'https://example.com/api/v1/topologies/host/replace' \
--header 'Content-Type: application/json; charset=UTF-8' \
-u phil:password123 \
-d '{"name":"host_new","active":true}'
```

In this example, if there is no topology named "host" the "host_new" topology is renamed "host".

Response example

Example response returning the newly replaced topology:

```
[
    {
        "name": "host_new",
        "description": "Host-based topology",
        "active": true
    }
]
```

Introduction to Graze API

Command Line Utility

Alert Analyzer Utility

The Alert Analyzer utility is a standalone process. It uses Natural Language Processing (NLP) techniques to analyze inbound event data. The Alert Analyzer divides text fields within the events into tokens. Based on the frequency of these tokens appearing in other events, it assigns an entropy value to the tokens and to the alerts in Cisco Crosswork Situation Manager.

See Entropy for more information on how Cisco Crosswork Situation Manager evaluates entropy and uses entropy thresholds to reduce the level of 'noise' from incoming event data.Entropy

See Configure Entropy Generation Schedule and Configure Entropy Thresholds for information on how to use Alert Analyzer features in the Cisco Crosswork Situation Manager UI.

Natural language processing analysis

The Alert Analyzer utility performs a number of linguistic analyses on events entering Cisco Crosswork Situation Manager. It then uses this linguistic analysis to calculate an entropy value for each token and then for every alert. See Entropy for more information.

Tokenization of text

The Alert Analyzer splits a text string at word boundaries, such as spaces or punctuation marks, into blocks. Each block of text is known as a token. For example, the following description has five tokens:

```
Link down on port 2/32
```

Token type identification

Commonly used word boundaries are often integral to the meaning of a token, for example, dots in IPV4 addresses. The Alert Analyzer identifies complete tokens of the following types within the structure of an event:

- IP addresses:
  - v4
  - v6
- MAC addresses
- OIDs
- Dates: Most standard formats.
- Numbers:
  - Integers
  - Real numbers
  - With and without unit suffixes, for example, 99%, 12kb, 345ms.
- File paths:
  - Forward slashes
  - Backward slashes
- GUIDs
- Hexadecimal numbers: With the 0x prefix.
- URLs
- Email addresses: Most standard formats.Identifying token types in arbitrary text is not an exact science and so, occasionally, the algorithms may identify tokens as a certain type which seems incorrect to a human.

After the Alert Analyzer has identified the token types, it can use them for masking and to identify tokens with high variation in a given alert.

Token masking

Tokens that change between events for the same alert can cause that alert to be assigned an incorrectly high entropy value. The most obvious example involves dates and times. If the description of an event is to be analyzed but each event contains a different timestamp, that timestamp will have a high entropy and skew the entropy for that alert as a whole. For other token types that change frequently, such as URLs or IP addresses, it may be desirable to retain the higher entropy associated with that token type because the changing value is significant.

You can configure the Alert Analyzer to include or exclude specific token types in the entropy analysis for each event partition.

You should consider masking dates, times and numbers from the entropy calculation.

Language processing techniques

The Alert Analyzer uses many standard techniques in language processing:

1. Case folding
    - Tokens that differ only by case, for example, 'WORD', 'Word' or 'word', are converted to the same case and considered equal.
    - Case folding is applied to all token types.
2. Stop words
    - You can add common or meaningless words, such as 'a', 'be', 'not', to a stop words file so that they are removed from the entropy calculation.
    - You can define a universal 'length' parameter so that any word at or below a certain length is treated as a stop word. For example, if set to '2', any words of one or two characters are ignored.
    - Stop words are applied to all token types.
3. Stemming
    - A technique used to reduce a word to its root to remove plurals or different tenses in verbs. Words with the same root are considered equal.
    - Note that some words, when stemmed, look unusual. For example, 'priority', 'priorities', prioritize, get stemmed to 'priorit'.
    - If stemming is enabled, the stemmed form is stored in the reference database.
    - Stemming is only applied to tokens of type 'word', that is, it is not applied to numbers, GUIDs, IP addresses, etc.

Priority words

Priority words are similar in concept to stop words but, rather than removing that word from the analysis as occurs with stop words, a priority word is assigned an entropy value of 1. For example, if ‘reboot’ is defined as a priority word, any tokens containing the word ‘reboot’ are given an entropy value of 1 regardless of how frequently the word appears in events.

1. Priority words are analyzed after stop words. If a token satisfies the criteria of a stop word, it is removed from the analysis and so cannot subsequently be considered as a priority word.
2. The reference database contains the calculated entropies for all tokens regardless of whether they are classed as priority words.

Partition-based analysis

You can configure the Alert Analyzer so that it calculates the entropy values for events for different partitions. As an example, you may want to run separate entropy calculations for different regions. In

this type of configuration, the same token can be given multiple entropy values within the same Moogfarmd deployment based on its frequency in the events within each partition. You can set up different configuration options for the different partitions. For example, in a particular partition, IP addresses may be masked whilst for another partition that may be unnecessary. In general, if a deployment uses the **"pre-partition"** method in Moogfarmd, that deployment benefits from partition-based entropy calculations.

See Configure Entropy Generation Schedule for further information on the Graze API endpoints you can use to configure partitions in the Alert Analyzer.

Alert Builder Reference

This is a reference for the Alert Builder Moolet.Alert Builder

You can change the behavior of the Alert Builder by editing the configuration properties in the **$MOOGSOFT_HOME/config/moolets/alert_builder.conf** configuration file. It contains the following properties:

name

Name of the Alert Builder Moolet. Do not change.

Type	String
Required	Yes
Default	**"AlertBuilder"**

classname

Moolet class name. Do not change.

Type	String
Required	Yes
Default	**4"CAlertBuilder"**

run_on_startup

Determines whether the Alert Builder runs when Cisco Crosswork Situation Manager starts. By default, it is set to true, so that when Moogfarmd starts, it automatically creates an instance of the Alert Builder. In this case you can stop it using **farmd_ctrl**.

Type	Boolean
Required	Yes
Default	**true**

moobot

Specifies a JavaScript file found in **$MOOGSOFT_HOME/moobots**, which defines the Alert Builder Moobot, which creates alerts.

Type	String
Required	Yes
Default	**AlertBuilder.js**

metric_path_moolet

Determines whether or not Cisco Crosswork Situation Manager includes the Alert Builder in the Event Processing metric for Self Monitoring.Self Monitoring

Type	Boolean
Required	Yes
Default	**true**

event_streams

A list of event streams, which the Alert Builder Moolet processes in this instance of Moogfarmd. The LAMs can be configured to send events on different streams. Moogfarmd, as specified in the Alert Builder configuration, then decides whether or not to process them. If Cisco Crosswork Situation Manager runs multiple Moogfarmds, you can have different event streams being processed by different Alert Builder Moolets.

You can comment out **event_streams**, or provide an empty list. Then, the Alert Builder processes every event that is published on the default **/Events** topic on the Message Bus.

You configure the Alert Builder Moolet by giving it a list of strings, for example, [“App A”, “App B”]. The result is that the Alert Builder listens for events published on **/Events/AppA**, and **/Events/AppB**, and processes that data. Importantly, in this example, events published to **/Events** or any other stream are ignored. You can have Moogfarmds that process completely separate event streams, or, multiple Moogfarmds that process some different event streams and some common event streams. You would do this when some of the alerts are common to all the applications that are being processed, but some are specific only to a given application. In this way, you can cluster alerts separately for each application by configuring the Sigalisers to only processes alerts from a specific upstream Alert Builder Moolet.

For example, if you have two separate applications that share the same network infrastructure: in Moogfarmd 1, you can have as the event streams, application A and networks, and, in Moogfarmd 2, you can have application B and networks. With this configuration, you can detect alerts and then create Situations that are relevant for just application A and similarly just for application B; however, if there is common networking infrastructure and problems occur with network failures across applications A and B, the Alert Builder can cluster these into Situations.

Type	String
Required	No
Default	**["AppA"]**

threads

Specifies the number of threads in the Alert Builder. Choose a value to match the event rate experienced by your system that allows time for alert creation.

Type	String
Required	Yes
Default	**4**

events_analyser_config

Allows you to specify a different Events Analyser configuration, for tokenizing and analysis rules, for each Alert Builder Moolet. If no configuration file is specified, the system default **`events_analyser.conf`** is used.

Type	String
Required	No
Default	**`"events_analyser.conf"`**

priming_stream_name

Stream name under which the Events Analyser runs in order to calculate token and alert entropies. If set to **`null`**, all alerts from all streams are included in the entropy calculations.

Type	String
Required	Yes
Default	**`null`**

priming_stream_from_topic

If set to **`true`**, Moogfarmd extracts the priming stream name from the event's stream. If set to **`false`**, Moogfarmd uses the stream configured in **`priming_stream_name`**.

Type	Boolean
Required	Yes
Default	**`false`**

Archiver Utility Command Reference

The archiver utility **`moog_archiver`** is a command line utility to archive and delete Situations, alerts and statistical data.

The utility is located at **`$MOOGSOFT_HOME/bin/utils/moog_archiver`**.

See Archive Situation and Alerts for more information.

Usage

```
moog_archiver [ --alert_filter <filter name> ] [ --update_batch_size <number of
Situations/rows> ] [ --delimiter <delimiter> ] [ --export ] [ --file_age <number
of days > ] [ --log_level WARN|INFO|DEBUG|TRACE ] [ --situation_filter ] [ --
loose_alert_age <number of days> ] [ --logconsole ] [ --logfilename ] [ --
include_statistics ] [ --statistics_age <number of days> ] [ --archive_path
<path> ] [ --remove ] [ --situation_age <number of days> ] [ --loose_alerts_only
] [ --delay_time <number of milliseconds> ] [ --id_batch_size <number of rows> ]
--help
```

Argument	Input	Description
`-a`, **`--alert_filter`**	String **`<filter name>`**	Include all loose alerts that match the specified global alert filter. Does not apply to alerts within Situations that are being archived.
`-b`, **`--`**	Integer **`<number of`**	Maximum number of alert/Situation rows to process at once during the export/deletion process.

update_batch_size	**Situations/rows>**	Increasing this value can speed up archiving but places more load on the database. Default is 1000.
-d, **--delimiter**	String **<delimiter>**	Delimiter to insert between values in the export file. Defaults to comma ",".
-e, **--export**	-	Export the archived data to a file.
-f, **--file_age**	Integer **<number of days>**	Delete files from the default directory **/usr/local/archived** that are older than the specified number of days.
-g, **--log_level**	String, one of **WARN | INFO | DEBUG | TRACE**	Log level controlling the amount of information logged by the utility. Default is WARN.
-i, **--situation_filter**	-	Include all Situations that match the specified global alert filter.
-l, **--loose_alert_age**	Integer **<number of days>**	Export data related to loose alerts older than the specified number of days. Default is 395.
--logconsole	-	Write logs to the console only.
--logfilename	String **<filename>**	Specify a log filename.
-m, **--include_statistics**	-	Include the deletion of statistical data. Statistical data can only be deleted, not archived.
-n, **--statistics_age**	Integer **<number of days>**	Delete statistical data older than the specified number of days. Default is 395.
-p, **--archive_path**	String **<path>**	Destination path for the archived data. Default is **/usr/local/archived**.
-r, **--remove**	-	Delete data from the database.
-s, **--situation_age**	Integer **<number of days>**	Include Situation data (and alerts within Situations) older than the specified number of days. Default is 395.
-t, **--loose_alerts_only**	-	Archive loose alerts only. Cannot be used with **-i** or **-s**.
-y, **--delay_time**	Integer **<number of milliseconds>**	Length of the delay between each batch operation. Can be used to slow the speed of archiving to reduce load on the database. Default is 0.
-z, **--id_batch_size**	Integer **<number of rows>**	Maximum number of rows to process per batch during the export/deletion process. Increasing this value can speed up archiving but places more load on the database. Default is 100.
-h, **--help**	-	Display the utility options and syntax.

Example

```
moog_archiver --export --remove --loose_alert_age 2 --situation_age 7 --
logfilename 10_12_20_archiver.log

Logging to : /var/log/moogsoft/10_12_20_archiver.log
```

Topology Loader Utility Command Reference

The Topology Loader utility **topology_loader** is a command line utility to add nodes, links and optional link descriptions to an existing topology.

The utility is located at **$MOOGSOFT_HOME/bin**.

See Load a Topology for more information.

Usage

```
topology_loader [-h] [-b=BATCH_SIZE] [--credentials=USERNAME:PASSWORD] -
f=CSV_FILE [--hostname=HOSTNAME] -t=TOPOLOGY
```

Argument	Input	Description
-b, **--batch-size**	Integer <**batch size**>	Number of links to create in each batch. Default is 10,000.
--credentials	String <**username:password**>	Graze API username and password. Default is graze:graze.
-f, **--file**	String <**csv_filename**>	Name of the .csv file containing the pairs of connected nodes with optional link descriptions.
--hostname	String <**hostname**>	Name of your Cisco Crosswork Situation Manager host running Nginx. Default is localhost.
-t, **--topology**	String <**topology name**>	Name of the topology in which to create the nodes and links. You must create the topology before you run the utility.
-h, **--help**	-	Display the utility options and syntax.

Example

An example to load nodes and links from the file "physical_topology.csv" into the "physical" topology on host "example.com" in batches of 15,000 is as follows:

```
topology_loader -b=15000 --credentials=phil:password123 -f=physical_topology.csv
--hostname=example.com -t=physical
```

The utility produces the following example output:

```
Executing:  topology_loader

INFO : [main][20200320 16:35:21.308 +0000] [CTopologyLoader.java:97] +|Starting,
reading from: physical_topology.csv|+
INFO : [main][20200320 16:35:21.348 +0000] [CCertificateProvider.java:83]
+|Found ssl_certificate setting: ssl_certificate
/etc/nginx/ssl/certificate.pem;|+
INFO : [main][20200320 16:35:21.352 +0000] [CTopologyProcessorHelper.java:66]
+|Using certificate: /etc/nginx/ssl/certificate.pem|+
INFO : [main][20200320 16:35:22.074 +0000] [CTopologyLoader.java:105] +|Read 10
link(s) for topology physical, processing in batches of 15,000|+
```

```
INFO : [main][20200320 16:35:23.067 +0000] [CTopologyProcessor.java:196] +|--
[204: https://example.com/api/v1/topologies/physical/links]|+
INFO : [main][20200320 16:35:23.068 +0000] [CTopologyProcessor.java:144] +|10 of
10 sent (100%)|+
INFO : [main][20200320 16:35:23.068 +0000] [CTopologyLoader.java:109]
+|Finished|+
```

Component Configuration

System Configuration

You can configure the various components of Cisco Crosswork Situation Manager using the system configuration file.

Configure your system

Edit the configuration file to control the behavior of the different components in your Cisco Crosswork Situation Manager system. You can find the file at **`$MOOGSOFT_HOME/config/system.conf`**.

See the System Configuration Reference for a full description of all properties. Some properties in the file are commented out by default. Uncomment properties to configure and enable them.

Message Bus

You can edit your Message Bus and RabbitMQ configuration in the **`mooms`** section of the file. It allows you to:

- Configure your Message Bus zones and brokers.
- Control and minimize message loss during a failure.
- Control how senders handle Message Bus failures.
- Control what happens during periods of extended Message Bus unavailability.
- Configure the SSL protocol you want to use.
- Specify the number of connections to use for each message sender pool.

For more information see the Message Bus documentation.

Database

You can edit your database configuration in the **`mysql`** section of the file:

- Configure your host name, database names and database credentials:

    1. host: Name of your host.
    2. moogdb_database_name: Name of the Moogdb database.
    3. referencedb_database_name: Name of the Cisco Crosswork Situation Manager reference database.
    4. intdb_database_name: Name of the Cisco Crosswork Situation Manager integrations database.

5. username:Username for the MySQL user that accesses the database.
6. encrypted_password: Encrypted password for the MySQL user.
7. password: Password for the MySQL user.
8. port: Default port that Cisco Crosswork Situation Manager uses to connect to MySQL.

- Configure the port, deadlock retry attempts and multi-host connections:
  - maxRetries: Maximum number of retries in the event of a MySQL deadlock.
  - retryWait: Number of milliseconds to wait between each retry attempt.
  - failover_connections: Hosts and ports for the different servers that are connected to the main host.
- Configure the SSL connections to the MySQL database:
  - trustStorePath: Path to location that stores the server certificate.
  - trustStoreEncryptedPassword: Path to location that stores your encrypted trustStore password.
  - trustStorePassword: Path to location that stores your trustStore password.

Elasticsearch

You can edit your search configuration in the **search** section of the file:

- Configure the Elasticsearch connection timeouts:
  - connection_timeout: Length of time in milliseconds before the connection times out.
  - request_timeout: Length of time in milliseconds before the request times out.
- Configure the Elasticsearch limit and nodes:
  - refresh_interval: Defines how often an Elasticsearch index refreshes. A newly indexed document is not visible in search results until the next time the index refreshes. Default is 30 seconds.
  - limit: Maximum number of search results that Elasticsearch returns from a search query.
  - nodes: Hosts and ports for the Elasticsearch servers connected in a cluster.

Failover

You can edit failover configuration in the **failover** section of the file:

- Configure persistence in the event of a failover:
  - persist_state: Enable or disable the persistence of the state of all Moolets in the event of a failover.
- Configure the Hazelcast cluster, this is Cisco Crosswork Situation Manager implementation of persistence:
  - network_port: Port to connect to on each specified host.
  - auto_increment: Enable for Hazelcast to attempt to the next incremental available port number if the configured port is unavailable.

  - hosts: List of hosts that can participate in the cluster.
  - man_center: Configures the cluster information that you can view in the Hazelcast Management Center UI.
  - cluster_per_group: Enable the stateful information from each process group to persist in a dedicated Hazelcast cluster.

- Configure failover options that apply to Moogfarmd and the LAMs:
  - keepalive_interval: Time interval in seconds at which processes report their active/passive status and check statuses of other processes.
  - margin: Amount of time in seconds after **keepalive_interval**before Cisco Crosswork Situation Manager considers processes that do not report their status to be dead.
  - failover_timeout: Number of seconds to wait for previously active process to become passive during a manual failover.
  - automatic_failover: Allow a passive process to automatically become active if no other active processes are detected in the same process group.
  - heartbeat_failover_after: Number of consecutive heartbeats that a process fails to send before Moogfarmd considers it inactive.

Process Monitor

You can edit the process monitor configuration in the **process_monitor** section of the file:

- Configure the heartbeat interval and delay:
  - heartbeat: Interval in milliseconds between heartbeats sent by processes.
  - max_heartbeat_delay: Number of milliseconds to wait before declaring heartbeat as missing.
- Configure the Moogfarmd and which processes you can control from the UI:

  1. group: Name of the group of processes and subcomponent processes that you want to control from the UI.
  2. instance: Name of the instance of Cisco Crosswork Situation Manager you want to configure.
  3. service_name: Name of the service you want to control.
  4. process_type: Type of process you want to control.
  5. reserved: Determines if Cisco Crosswork Situation Manager considers the process as critical in process monitoring.

Encryption

You can edit the encryption configuration in the **encryption** section of the file:

- encryption_key_file: Default location of the encryption key file.

High Availability

You can edit the high availability configuration in the **ha** section of the file.

- cluster: Default HA cluster name

Port ranges

You can edit the port range that Cisco Crosswork Situation Manager services use when they look for open ports.

1. port_range_min: Minimum port number in the range.
2. port_range_max: Maximum port number in the range.

Example

The following example shows **`system.conf`** with the default configuration and all available properties enabled:

```
{
   "mooms": {
      "zone": "",
      "brokers": [{
         "host": "localhost",
          "port": 5672
      }],
      "username": "moogsoft",
      "password": "m00gs0ft",
      "encrypted_password": "e5uO0LY3HQJZCltG/caUnVbxVN4hImm4gIOpb4rwpF4=",
      "threads": 10,
      "message_persistence": false,
      "message_prefetch": 100,
      "max_retries": 100,
      "retry_interval": 200,
      "cache_on_failure": false,
      "cache_ttl": 900,
      "connections_per_producer_pool": 2,
      "confirmation_timeout": 2000,
      "ssl": {
         "ssl_protocol": "TLSv1.2",
         "server_cert_file": "server.pem",
         "client_cert_file": "client.pem",
         "client_key_file": "client.key"
      }
   },
   "mysql": {
      "host": "localhost",
      "moogdb_database_name": "moogdb",
      "referencedb_database_name": "moog_reference",
      "intdb_database_name": "moog_intdb",
      "username": "ermintrude",
      "encrypted_password": "vQj7/yom7e5ensSEb10v2Rb/pgkaPK/4OcUlEjYNtQU=",
      "password": "m00",
      "port": 3306,
      "maxRetries": 10,
      "retryWait": 50,
      "failover_connections": [
         {
            "host": "193.221.20.24",
            "port": 3306
         },
         {
             "host": "143.47.254.88",
```

```
                "port": 3306
            },
            {
                "host": "234.118.117.132",
                "port": 3306
            }
        ],
        "ssl": {
            "trustStorePath": "etc/truststore",
            "trustStoreEncryptedPassword":
"vQj7/yom7e5ensSEb10v2Rb/pgkaPK/4OcUlEjYNtQU=",
            "trustStorePassword": "moogsoft"
        }
    },
    "search": {
        "connection_timeout": 1000,
        "request_timeout": 10000,
        "refresh_interval": 30,
        "limit": 1000,
        "nodes": [{
            "host": "localhost",
            "port": 9200
        }]
    },
    "failover": {
        "persist_state": false,
        "hazelcast": {
            "network_port": 5701,
            "auto_increment": true,
            "hosts": ["localhost"],
            "man_center":
            {
                "enabled": false,
                "host": "localhost",
                "port": 8091
            },
            "cluster_per_group": false
        },
        "keepalive_interval": 5,
        "margin": 10,
        "failover_timeout": 10,
        "automatic_failover": false,
        "heartbeat_failover_after": 2
    },
    "process_monitor": {
        "heartbeat": 10000,
        "max_heartbeat_delay": 1000,
        "processes": [{
            "group": "moog_farmd",
            "instance": "",
            "service_name": "moogfarmd",
            "process_type": "moog_farmd",
            "reserved": true,
            "subcomponents": [
                "AlertBuilder",
                "Default Cookbook",
                "TeamsMgr",
```

```
            "Housekeeper",
            "AlertRulesEngine",
            "SituationMgr",
            "Notifier"
            ]},
            {
                "group": "servlets",
                "instance": "",
                "service_name": "apache-tomcat",
                "process_type": "servlets",
                "reserved": true,
                "subcomponents": [
                    "moogsvr",
                    "moogpoller",
                    "toolrunner",
                    "situation_similarity"
                ]},
            {
                "group": "logfile_lam",
                "instance": "",
                "service_name": "logfilelamd",
                "process_type": "LAM",
                "reserved": false
            },
            {
                "group": "rest_lam",
                "instance": "",
                "service_name": "restlamd",
                "process_type": "LAM",
                "reserved": false
            },
            {
                "group": "socket_lam",
                "instance": "",
                "service_name": "socketlamd",
                "process_type": "LAM",
                "reserved": false
            },
            {
                "group": "trapd_lam",
                "instance": "",
                "service_name": "trapdlamd",
                "process_type": "LAM",
                "reserved": false
            },
            {
                "group": "rest_client_lam",
                "instance": "",
                "service_name": "restclientlamd",
                "process_type": "LAM",
                "reserved": false
            }
        ]
},
"encryption": {
   "encryption_key_file": "/location/of/.key"
},
"ha": {
   "cluster": "MOO"
},
"port_range_min": 50000,
```

```
    "port_range_max": 51000
}
```

Start and stop Moogfarmd

Restart the Moogfarmd service to activate any changes you make to the system configuration file.

The service name is **moogfarmd**.

See Control Moogsoft AIOps Processes for further details.Control Processes

System Configuration Reference

This is a reference for the system configuration file located at **$MOOGSOFT_HOME/config/system.conf**. It contains the following sections and properties:

Message Bus (MooMs)

connections_per_producer_pool

The number of connections to use for each message sender pool. For example, if a message sender pool has 20 channels and this property is set to 2, the channels are split across both connections so that each has 10 channels. To configure this property, you must manually add it to the **mooms** section.

Type	Integer
Required	No
Default	2

zone

Name of the zone.

Type	String
Required	No
Default	N/A

brokers

Hostname and port number of the RabbitMQ broker.

Type	Array
Required	No
Default	**"host" : "localhost", "port" : 5672**

username

Username of the RabbitMQ user. This needs to match the RabbitMQ broker configuration. If commented out, it uses the default "guest" user.

Type	String
Required	No
Default	**guest**

password

Password for the RabbitMQ user. You can choose to either have a password or an encrypted password, you cannot use both.

Type	String
Required	Yes. If you are not using **`encrypted password`**.
Default	**`guest`**

encrypted_password

Encrypted password for the RabbitMQ user. You can choose to either have a password or an encrypted password, you cannot use both. See Moog Encryptor if you want to encrypt your password.

Type	String
Required	Yes. If you are not using **`password`**.
Default	N/A

threads

Number of threads a process can create in order to consume the messages from the Message Bus. If not specified, the thread limit = (Number of processors x 2) + 1. Altering this limit affects the performance of Cisco Crosswork Situation Manager processes such as Moogfarmd and Moogpoller.

If your logs indicate an issue in creating threads, Cisco advises that you increase the ulimit, the maximum number of file descriptors each process can use, for the Cisco Crosswork Situation Manager user. You can set this limit in **`/etc/security/limits.conf`**.

Type	Integer
Required	No
Default	**`10`**

message_persistance

Controls whether RabbitMQ persists important messages. Message queues are durable by default and data is replicated between nodes in High Availability mode. Setting this value to **`false`** means that replicated data is not stored to disk.

Type	Boolean
Required	No
Default	**`true`**

message_prefetch

Controls how many messages a process can take from the Message Bus and store in memory as a buffer for processing. This configuration allows processes to regulate message consumption which can ease backlog and memory consumption issues. The higher the number, the more messages held in the process's memory. Set to 0 for unlimited processing. To achieve high availability of messages and ensure messages are processed, the value of this should be higher than 0.

Type	Integer
Required	No
Default	**`0`**

max_retries

Maximum number of attempts to resend a message that failed to send. Cisco Crosswork Situation Manager only attempts a retry when there is a network outage or if **cache_on_failure** is enabled.

You can use this in conjunction with the **retry_interval** property. For example, a combination of 100 maximum retries and 200 milliseconds for retry interval leads to a total of 20 seconds. The combined default value for these properties was chosen to handle the typical time for a broker failover in a clustered environment.

Type	Integer
Required	No
Default	**100**

retry_interval

Maximum length of time to wait in milliseconds between each attempt to retry and send a message that failed to send.

You can use this in conjunction with the **max_retries** property. The combined value for these properties was chosen to handle the typical time for broker failover in a clustered environment.

Type	Integer
Required	No
Default	**200**

cache_on_failure

Controls whether Cisco Crosswork Situation Manager caches the message internally and resends it if there is an initial retry failure. The system attempts to resend any cached messages in the order they were cached until the time-to-live value, defined by the **cache_ttl** property, is reached.

Type	Boolean
Required	No
Default	**false**

cache_ttl

Length of time in seconds that Cisco Crosswork Situation Manager keeps cached messages in the cache list before discarding them. If a message is not successfully resent within this timeframe it is still discarded.

This defaults to 900 seconds (15 minutes). Increasing this value has a direct impact on sender process memory.

Type	Integer
Required	No
Default	**900**

confirmation_timeout

Length of time in milliseconds to wait for the Message Bus to confirm that a broker has received a message. Cisco does not advise changing this value.

Type	Integer
Required	No
Default	**2000**

Message Bus SSL

ssl_protocol

SSL protocol you want to use. JRE 8 supports "TLSv1.2", "TLSv1.1", "TLSv1" or "SSLv3".

Type	String
Required	No
Default	**TLSv1.2**

server_cert_file

Path to the directory that contains the SSL certificates. You can use a relative path based upon the **$MOOGSOFT_HOME** directory. For example, **config** indicates **$MOOGSOFT_HOME/config**.

Type	String
Required	No
Default	**server.pem**

client_cert_file

Enables client authentication if you provide a client certificate and key file.

Type	String
Required	No
Default	**client.pem**

client_key_file

Enables client authentication if you provide a client key file. The file must be in PKCS#8 format.

Type	String
Required	No
Default	**client.key**

MySQL

host

Host name or server name of the server that is running MySQL.

Type	String
Required	No
Default	**localhost**

moogdb_database_name

Name of the primary Cisco Crosswork Situation Manager database.

Type	String
Required	No
Default	**moogdb**

referencedb_database_name

Name of the Cisco Crosswork Situation Manager reference database.

Type	String
Required	No
Default	**moog_reference**

intdb_database_name

Name of the integrations database.

Type	String
Required	No
Default	moog_intdb

username

Username of the MySQL user.

Type	String
Required	No
Default	**ermintrude**

password

Password for the MySQL user. You can choose to either have a password or an encrypted password, you cannot use both.

Type	String
Required	Yes, if you are not using **encrypted password**.
Default	**m00**

encrypted_password

Encrypted password for the MySQL user. You can choose to either have a password or an encrypted password, you cannot use both. See Moog Encryptor if you want to encrypt your password.

Type	String
Required	Yes, if you are not using **password**.
Default	N/A

port

Port that MySQL uses.

Type	Integer

Required	No
Default	**3306**

maxRetries

Maximum number of MySQL query retries to attempt in the event of a deadlock.

Type	Integer
Required	No
Default	**10**

retryWait

Length of time in milliseconds to wait between retry attempts.

Type	Integer
Required	No
Default	**50**

failover_connections

Hosts and ports for the different servers that are connected to the main host. For example, primary-primary, primary-secondary. In the event of connection failover, the connection cannot be read-only (secondary).

Type	List
Required	No
Default	N/A

MySQL SSL

trustStorePath

Path to tNohe directory that contains the trustStore you want to use for SSL connections to your MySQL database. You can use a relative path based upon the **$MOOGSOFT_HOME** directory. For example, **config** indicates **$MOOGSOFT_HOME/config/truststore.**

Type	String
Required	No
Default	**etc/truststore**

trustStoreEncryptedPassword

Your encrypted trustStore password. You can choose to either have a password or an encrypted password, you cannot use both. See Moog Encryptor if you want to encrypt your password.

Type	String
Required	Yes, if you are not using **trustStorePassword**.
Default	N/A

trustStorePassword

Your trustStore password. You can choose to either have a password or an encrypted password, you cannot use both.

Type	String
Required	No, if you are not using **`trustStoreEncryptedPassword`**.
Default	**`moogsoft`**

connection_timeout

Length of time in milliseconds before the connection to the Elasticsearch server times out.

Type	Integer
Required	No
Default	**`1000`**

nodes

Hosts and ports for the different Elasticsearch servers connected in a cluster.

Type	Array
Required	No
Default	**`"host" : "localhost",`** **`"port" : 9200`**

Failover

persist_state

Enable or disable the persistence of the state of all Moolets in the event of a failover.

Type	Boolean
Required	No
Default	**`false`**

network_port

Port to connect to on each specified host in your Hazelcast cluster.

Type	Integer
Required	No
Default	**`5701`**

auto_increment

Enable for Hazelcast to attempt to connect to the next incremental available port number if the configured port is unavailable.

Type	Boolean
Required	No
Default	**`true`**

hosts

List of hosts that can participate in the cluster.

Type	Array
Required	No
Default	**localhost**

man_center

Specifies the cluster information that you can view in the Hazelcast Management Center UI.

Type	List
Required	No
Default	**"enabled" : false,** **"host" : "localhost",** **"port" : 8091**

cluster_per_group

Enable the stateful information from each process group to persist in a dedicated Hazelcast cluster.

Type	Boolean
Required	No
Default	**false**

Moogfarmd Failover

keepalive_internal

Time interval in seconds at which processes report their active or passive status and check statuses of other processes.

Type	Integer
Required	No
Default	**5**

margin

Amount of time in seconds after **keepalive_interval** before Cisco Crosswork Situation Manager considers processes that do not re_port their status to be dead.

Type	Integer
Required	No
Default	**10**

failover_timeout

Amount of time in seconds to wait for previously active process to become passive during manual failover.

Type	Integer
Required	No
Default	**10**

automatic_failover

Allow a passive process to automatically become active if no other active processes are detected in the same process group.

Type	Boolean
Required	No
Default	**false**

Process Monitor

heartbeat

Interval in milliseconds between heartbeats sent by processes.

Type	Integer
Required	Yes
Default	10000

max_heartbeat_delay

Number of milliseconds to wait before declaring heartbeat as missing. Defaults to 10% of the heartbeat.

Type	Integer
Required	No
Default	**1000**

Processes

Groups of processes that you want to be able to stop, start and restart from Self Monitoring in the Cisco Crosswork Situation Manager UI. For each group you can configure the following options:

group

Name of the process group that Cisco Crosswork Situation Manager uses when it starts and stops the service.

Type	String
Required	Yes
Default	**N/A**

instance

Name of the instance for the process.

Type	String
Required	Yes
Default	**N/A**

display_name

Additional identification label that appears in the UI.

Type	String
Required	No
Default	N/A

cluster

Name of the process's cluster. This overrides the default cluster for a process. If left empty, the Cisco Crosswork Situation Manager uses the process's default cluster.

Type	String
Required	No
Default	N/A

service_name

Name of the service script that Cisco Crosswork Situation Manager uses to control the process. If you do not configure a service name, Cisco Crosswork Situation Manager uses the group name, removing underscores and appending a 'd'. For example, "traplam" becomes "traplamd".

Type	String
Required	No
Default	N/A

process_type

Type of process. If left empty, Cisco Crosswork Situation Manager calculates the type based on the group name.

Type	String
Required	No
Default	N/A
Valid Values	**`moog_farmd`**, **`servlet`**, **`LAM`**

reserved

Determines if the process produces a warning in the UI when it is running. Processes that are unreserved do not produce a warning.

Type	Boolean
Required	No
Default	**`true`**

subcomponents

Specifies which Moolets are reserved for the Moogfarmd process. If left empty, no Moolets are reserved for the Moogfarmd process.

Type	Array
Required	No
Default	N/A

Encryption

encryption_key_file

Default location of the encryption key file.

Type	String
Required	No
Default	**`/location/of/.key`**

High Availability (HA)

cluster

Default HA cluster name.

Type	String
Required	No
Default	**`MOO`**

Port Range

port_range_min

Minimum port number in the range that the Cisco Crosswork Situation Manager services use when they look for open ports.

Type	String
Required	No
Default	**`50000`**

port_range_max

Maximum port number in the range that the Cisco Crosswork Situation Manager services use when they look for open ports.

Type	String
Required	No
Default	**`51000`**

Security Configuration Reference

This is a reference for security configuration in Cisco Crosswork Situation Manager. You can edit **`$MOOGSOFT_HOME/config/security.conf`** to configure security features such as LDAP and SAML.Configure Single Sign-On with LDAPConfigure Single Sign-On with SAML

LDAP Connection Properties

You can configure the LDAP connection using the following properties:

url

The protocol (LDAP or LDAPS) along with the host and port of your LDAP server. For example: ldap://172.16.124.169:389.

Type: String

Required: Yes

Default: N/A

connectionTimeout

Defines the connection timeout in milliseconds.

Type: String

Required: Yes

Default: **`30000`**

readTimeout

Defines the read timeout in milliseconds.

Type: String

Required: Yes

Default: **`30000`**

predefinedUser

If enabled, the user account information must exist in the local database as well as the LDAP server and predefined user details are used to populate created or updated user accounts.

If disabled, Cisco Crosswork Situation Manager creates or updates user accounts with the LDAP information.

Type: String

Required: Yes

Default: False

LDAP Attribute Search Properties

You can configure the authentication bind, DN resolution method and attribute search with the following properties:

resolutionType

Defines the method to look up the DN (Distinguished Name), a unique path to any object in the active directory.

Type: String

Required: Yes

One of: **`direct, lookup`**

Default: N/A

There are two methods to choose from:

- direct: If using this method, the user DN is created using the **usernameAttribute** and **userDnPostfix** properties. These properties are required. For example:

```
"userDnResolution": {
        "resolutionType" : "direct",
                "direct" : {
                        "usernameAttribute": "uid",
                "userDnPostfix": "ou=People,dc=moogsoft,dc=com"
                }
},
```

For a user called John Smith, the user DN is:

```
uid=john.smith,ou=People,dc=moogsoft,dc=com
```

- lookup: If using this method, Cisco Crosswork Situation Manager searches for the user in the LDAP server using a combination of **usernameAttribute** and **userBaseSearchFilter** as a filter and **userBaseDn** as a base to find the DN. These properties are required. For example:

```
"userDnResolution": {
        "resolutionType" : "lookup",
                "lookup" :    {
                        "usernameAttribute": "sAMAccountName",
            "userBaseDn" : "ou=People,dc=moogsoft,dc=com",
            "userBaseSearchFilter" : "(objectclass=person)",
                }
},
```

Optionally for both "direct" and "lookup" methods, you can use the **userDnLookupUser, userDnLookupPassword** and **encryptedUserDnLookupPassword**properties to define the user to look up each DN in your directory. See Moog Encryptor for more information if you want to use password encryption.

If you leave the **userDnLookupUser** property empty, LDAP uses the **systemUser** defined in the LDAP Group Search section instead.

attributeSearchFilter

Defines an optional LDAP attribute filter to search for user attributes.

Type: String

Required: No

Default: **(objectclass=*)**

attributeMap

Defines an attribute map between the LDAP user attributes and the user attributes in the Cisco Crosswork Situation Manager database.

Type: String

Required: No

Default: N/A

This property uses the format:

```
"attributeMap": {
       "db_column_5": "ldap_attribute_1",
       "db_column_2": "ldap_attribute_8",
       "db_column_3": "ldap_attribute_8",
}
```

LDAP Group Search and Mapping

You can configure the following properties in the LDAP group search section:

systemUser

Username of the system user to bind and search for user group information. LDAP uses this user if you leave the **`userDnLookupUser`** property empty. The system sends two bind requests and two search requests with LDAP. If you do not configure a system user, the user bind chosen for authentication is also used for the LDAP group search.

Type: String

Required: No

Default: N/A

systemPassword

Password of the system user to bind and search for user group information.

Type: String

Required: No

Default: N/A

groupBaseDn

DN for the part of the LDAP structure that contains the user groups. This is used in conjunction with the **`memberAttribute`** to find any LDAP groups the user belongs to. These groups are then mapped to a local role using the **`roleMap`** property.

Type: String

Required: No

Default: N/A

memberAttribute

Attribute used to look for group members.

Type: String

Required: No

Default: **`member`**

groupNameAttribute

Attribute used to look for group name.

Type: String

Required: No

Default: **`CN`**

roleMap

Defines the role mappings between the user directory and Cisco Crosswork Situation Manager.

Type: String

Required: No

Default: N/A

LDAP AssignTeams Properties

You can configure the following sub-properties of **`assignTeams`** to synchronize team assignment between the user directory and the teams in Cisco Crosswork Situation Manager.

assignTeams

Sychronizes team assignment between the user directory and the teams in Cisco Crosswork Situation Manager.

Type: String

Required: No

Default: N/A

teamMap

Defines the LDAP attribute or custom attribute that maps to team names in Cisco Crosswork Situation Manager. You can provide the mapping as a JSON object. For example:

```
{ "LDAP Team" : "Moogsoft Team", "Another LDAP Team" : "Another Moogsoft team" }
```

Type: JSON Object

Required: No

Default: N/A

useGroupName

Enable to use the LDAP group name as the team name in Cisco Crosswork Situation Manager.

Type: Boolean

Required: No

Default: **`false`**

createNewTeams

Creates a team or teams if they do not exist in Cisco Crosswork Situation Manager. If you leave **`teamMap`** empty, the teams adopt their LDAP teams names.

Type: Boolean

Required: No

Default: **`false`**

LDAP SSL Properties

You can optionally configure SSL to enable TLS authentication:

ssl_protocol

Defines the SSL protocol you want to use.

Type: String

Required: No

Default: **`TLSv1.2`**

server_cert_file

SSL server certificate.

Type: String

Required: No

Default: N/A

client_cert_file

SSL client certificate.

Type: String

Required: No

Default: N/A

client_key_file

Client key file.

Type: String

Required: No

Default: N/A

SAML Service Provider Properties

You can configure a SAML realm by giving it a name and editing the following properties:

idpMetadataFile

Location of the identity provider's metadata file. The metadata file provides information on how to connect to the IdP. Cisco Crosswork Situation Manager requires the file to be in .xml format.

Type: String

Required: Yes

Default: **"/usr/share/moogsoft/etc/saml/my_idp_metadata.xml"**

spMetadataFile

Location of the service provider's metadata file. Cisco Crosswork Situation Manager writes the SP metadata information to this file. This location must be accessible and editable by the Apache Tomcat user. Cisco Crosswork Situation Manager requires the file to be in .xml format. If your IdP does not have an SP metadata file generator, you can create one manually. See Build a Service Provider Metadata File for instructions.Build a Service Provider Metadata File

Type: String

Required: No

Default: **"/usr/share/moogsoft/etc/saml/my_sp_metadata.xml"**

defaultRoles

Default roles that Cisco Crosswork Situation Manager assigns to new users upon first login using SAML. If the user already has a role mapping, Cisco Crosswork Situation Manager uses that instead.

Type: Array

Required: Yes

Default: **["Operator"]**

defaultTeams

Default teams that Cisco Crosswork Situation Manager assigns to new users upon first login using SAML. You can create an empty list if you do not want to assign new users to a team.

Type: Array

Required: No

Default: **["Cloud DevOps"]**

defaultGroup

Default primary group that Cisco Crosswork Situation Manager assigns to new users upon first login using SAML.

Type: Array

Required: Yes

Default: **["End-User"]**

SAML User Mapping Properties

You can configure how to map IdP user fields to existing Cisco Crosswork Situation Manager users and how to map user fields for new users. All mappings are case sensitive. Each mapping follows the format:

"MoogsoftAttribute" : "IdPAttribute"

existingUserMappingField

Defines the field that Cisco Crosswork Situation Manager uses to map existing users to your IdP users.

Type: String

Required: No

One of: **`username, email`**

Default: **`"username"`**

username

Defines the IdP's attribute that maps to username in Cisco Crosswork Situation Manager.

Type: String

Required: Yes

Default: **`"$Email"`**

email

Defines the IdP's attribute that maps to email in Cisco Crosswork Situation Manager.

Type: String

Required: Yes

Default: **`"$Email"`**

fullname: Defines the IdP attributes that map to full name in Cisco Crosswork Situation Manager.

Type: String

Required: Yes

Default: **`"$FirstName $LastName"`**

SAML Optional Properties

You can customize your SAML realm with a number of optional properties:

contactNumber

Defines the IdP attribute that maps to contact number in Cisco Crosswork Situation Manager.

Type: String

Required: No

Default: **`"phone",`**

department

Defines the IdP attribute that maps to department in Cisco Crosswork Situation Manager.

Type: String

Required: No

Default: **`"department",`**

primaryGroup

Defines the IdP attribute that maps to primary group inCisco Crosswork Situation Manager.

Type: String

Required: No

Default: **"primaryGroup",**

timezone

Defines the IdP attribute that maps to timezone in Cisco Crosswork Situation Manager.

Type: String

Required: No

Default: **"timezone",**

SAML assignTeams Properties

You can configure the following sub-properties of **assignTeams** to synchronize team assignment between the SAML user directory and the teams in Cisco Crosswork Situation Manager:

teamAttribute

Defines the IdP attribute that maps to teams in Cisco Crosswork Situation Manager.

Type: String

Required: No

Default: "groups"

teamMap

Defines the IdP attribute or custom attribute that maps to team names in Cisco Crosswork Situation Manager.

Type: JSON Object

Required: No

Default: **{ "IdP Team" : "Moogsoft AIOps Team", "Another IdP Team" : "Another AIOps team" }**

createNewTeams

Creates a team or teams if they do not exist in Cisco Crosswork Situation Manager. If you leave **teamMap** empty, the teams adopt their IdP teams names.

Type: Boolean

Required: No

Default: **false**

SAML assignRoles Properties

roleAttribute

Defines the IdP attribute containing role information.

Type: String

Required: No

Default: **"groups"**

roleMap

Defines the IdP attribute that maps to Cisco Crosswork Situation Manager roles.

Type: JSON Object

Required: No

Default: **{ "IdP Standard User" : "Operator", "IdP Manager User" : "Manager" }**

SAML Security Properties

keystorePassword

Your unencrypted keystore password. Any whitespace in the name is replaced with an underscore.

Type: String

Required: No

Default: **"<my_realm>_secret"**

encryptedKeystorePassword

Your encrypted keystore password. Any whitespace in the name is replaced with an underscore. You can have either an unencrypted keystore password or an encrypted keystore password, but you cannot use both. See Moog Encryptor for more information on encrypting passwords.

Type: String

Required: No

Default: N/A

privateKeyPassword

Your private key password. Any whitespace in the name is replaced with an underscore.

Type: String

Required: No

Default: **"<my_realm>_secret"**

maximumAuthenticationLifetime

Maximum time in seconds for Cisco Crosswork Situation Manager to receive an IdP's SAML assertion before it becomes invalid.

Type: Integer

Required: No

Default: **2592000** (720 hours)

serviceProviderEntityId

Service Provider Entity ID assertion number. Some IdPs require this ID.

Type: String

Required: No

Default: **"MoogsoftAIOps"**

Service Provider Metadata Reference

This is a reference for Build a Service Provider Metadata File. Each SP metadata .xml file accepts the following elements:Build a Service Provider Metadata File

entityID

Unique identifier or name for the service provider. The ID should be a URN or a URL.

Type: String

Required: Yes

Example: **https://example.moogsoftaiops.com/moogsvr/mooms**

ID

Unique identifier for the root metadata element.

Type: String

Required: No

Example: **TW9vZ3NvZnRBSU9wcw==**

validUntil

Defines the expiration date of the metadata file. The date should be in ISO 8601 format.

Type: String

Required: No

Example: **2018-08-10T07:47:41+00:00**

AuthnRequestsSigned

If enabled, Cisco Crosswork Situation Manager signs SAML authentication requests as part of the Single Sign-On.

Type: Boolean

Required: No

Default: **false**

WantAssertionsSigned

If enabled, Cisco Crosswork Situation Manager expects IdPs to sign any SAML assertions it sends.

Type: String

Required: No

Default: **`false`**

KeyDescriptor

Defines the type of signing or the type of encryption that Cisco Crosswork Situation Manager uses.

Type: String

Required: No

One of: **`use = "signing", use = "encryption"`**

X509Certificate

Self-signed certificate that allows Cisco Crosswork Situation Manager to sign and encrypt each SAML assertion. The certificate should be in DER format and base-64 encoded.

Type: String

Required: No

Example: **`MIIDijCCAnICCQD[...]`**+6SBfDCrWFsw==

AssertionConsumerService

Defines the URL or endpoint that receives the SAML assertions. The Location is for the URL and the Binding identifies the method. Supported bindings include: HTTP-Artifact, HTTP-POST, HTTP-POST-SimpleSign, HTTP-Redirect and SOAP.

Type: String

Required: Yes

Example: **`Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST Location="https://localhost/moogsvr/mooms?request=samlResponse"`**

Moogfarmd and Core Data Processing

Moogfarmd is the core system application that runs all of the algorithms and automation relevant to Cisco Crosswork Situation Manager. It is responsible for the following:

- Creating alerts.
- Analyzing alerts to determine their significance.
- Clustering alerts into Situations.
- Performing automation relating to the automated response such as escalation, routing, notification, invitation of either alerts or Situations.

The topics in this guide help you configure the data processing components of Moogfarmd:

You can run one or many instances of Moogfarmd on your Cisco Crosswork Situation Manager system.

Services

The Cisco Crosswork Situation Manager installation installs Moogfarmd as a service:

```
/etc/init.d/moogfarmd
```

A backup Moogfarmd service script is located at **`$MOOGSOFT_HOME/etc/service-wrappers/moogfarmd`**.

If you run multiple instances of Moogfarmd on the same host, copy and modify the default Moogfarmd service script for each Moogfarmd running on the host:

- Copy **`$MOOGSOFT_HOME/etc/service-wrappers/moogfarmd`** to **`/etc/init.d/mymoogfarmd`**.
- Edit the following parameters in the **`/etc/init.d/mymoogfarmd`** file:

```
SERVICE_NAME=mymoogfarmd
CONFIG_FILE=$PROCESS_HOME/config/my_moog_farmd.conf
```

- You now have a new service to be used to start your own specific Moogfarmd:

```
service mymoogfarmd start
```

Learn More

For information on starting, stopping and configuring Moogfarmd, see the Moogfarmd Reference.

Moogfarmd Reference

Moogfarmd controls all other services in Cisco Crosswork Situation Manager and manages which algorithms and Moolets are running.

Services

We advise that you start Moogfarmd as a service. A service script is provided out of the box for the default Moogfarmd configuration and is located here:

```
/etc/init.d/moogfarmd
```

A backup Moogfarmd service script is located at **`$MOOGSOFT_HOME/etc/service-wrappers/moogfarmd`**.

If using multiple instances of Moogfarmd on the same host, we advise that you copy and modify the default Moogfarmd service script for each Moogfarmd running on the host.

Run the Moogfarmd Service Daemon

Moogfarmd is a command line executable that can be run as a service daemon.

To execute the daemon and view available arguments run:

```
moog_farmd --help
```

By default, you do not need either 'config' or 'instance'. If you run the system without configuring either of these, the moogfarmd instance loads the default configuration file for moogfarmd, and responds to farmd_ctrl with no instance specified. See High Availability Overview for more information on High Availability.

The **moog_farmd** command line executable accepts the following arguments:

Option	Input	Description
--clear_state	-	Clears any persisted state information associated with Moogfarmd on startup.
--cluster <arg>	String: <cluster name>	Name of the High Availability (HA) cluster. Overwrites the value in the configuration file.
--config <arg>	String: <file path/name>	Name and path of the configuration file specific to the running Moogfarmd instance.
--group <arg>	String: <group name>	Name of the HA group. Overwrites the value in the configuration file.
-h, --help	-	Displays all command line options.
--instance <arg>	String: <instance name>	Enables you to name the Moogfarmd instance. You can refer to this name in the farmd_ctrl utility, which allows you to start, restart and reload the various Moolets.
--logconsole	-	Instructs Moogfarmd to write logs to the console only.
--logfilename <arg>	String: <file path/name>	Name and path of the Moogfarmd log file.
-l, --loglevel <arg>	INFO|WARN|ALL	Specifies the debug level. Defaults to WARN, which is the recommended level in all production implementations.
--mode <arg>	String: active/passive	Starts the process in passive or active mode. The default is active.
--service_instance <arg>	String: <service suffix>	Suffix for the service name.
-v, --version	-	Displays the Moogfarmd version number.

Configuration

You can control Moogfarmd behavior through the following files:

- system.conf: the general Cisco Crosswork Situation Manager system configuration file is located in **$MOOGSOFT_HOME/config/system.conf**. See System Configuration.
- moog_farmd.conf: configuration specific to Moogfarmd operation. If you run multiple instances of Moogfarmd, each needs it own configuration file. All instances of Moogfarmd which do not specify a different --config use the default configuration file located in **$MOOGSOFT_HOME/config/moog_farmd.conf**.

Moogfarmd runs individual isolated applications called Moolets inside the Moogfarmd app container. Moolets are a parallel concept to servlets in a traditional enterprise application container such as Tomcat. Moogfarmd controls the flow of data through the Moolets where the data can come via the Message Bus or from other Moolets.

You can configure the following properties in the Moogfarmd configuration files:

alert_threshold

The minimum number of alerts that must be present in a cluster before it can become a Situation.

Type	Integer
Required	Yes
Default	2

bus_thread_pool_queue_limit

The maximum number of Message Bus messages to store in memory.

Type	Integer
Required	Yes
Default	0 (unlimited)

Note

If you reduce this value, message data may be lost.

db_connections

Specifies the number of database connections for Moogfarmd independently of the number of threads.

Type	Integer
Required	Yes
Default	30

Note

Do not change this setting count unless instructed by Cisco Support.

file_only_config

This setting serves two purposes related to the Congfiguration Migration Utility:

1. Setting the property to **`true`** before upgrading prevents the configuration migration utility from running.
2. Setting the property to **`true`** after upgrading causes Cisco to ignore all database configurations for Cookbook and Tempus clustering algorithms as well as merge groups, and only load their file configurations instead.

Type	Boolean
Required	No
Default	True

ha

The moog_farmd.conf file includes settings you can use to specify the **`cluster`**, **`group`**, and **`instance`** for an HA configuration hierarchy.

Note

Do not change any other HA settings in this file unless instructed by Cisco Support.

maximum_rest_requests

The maximum allowed number of concurrent asynchronous REST tasks. Increasing the value consumes more system resources.

Type	Integer
Required	Yes
Default	200

moobot_optimization

The optimization level to use for Moobots.

Type	Integer
Required	Yes
One of	-1: Moobots are interpreted. 0-9: Moobots are precompiled. 0 is minial optimization and 9 is maximum optimization. See Mozilla optimization documentation for more information.
Default	0

moolet_queue_size_limit

The maximum number of messages from each Moolet to store in memory. You can overwrite this setting in individual Moolet configurations.

Type	Integer
Required	Yes
Default	0 (unlimited)

Note

If you reduce this value, message data may be lost.

retention_period

Length of time in seconds to keep unchanged closed/superseded Situations in memory.

Type	Integer
Required	Yes
Default	86400 (1 day)

sig_resolution

Section of the file containing properties related to Situation resolution.

Type	Object
Required	Yes
Default	N/A

sig_similarity_limit

The percentage of alerts two Situations must share before they are merged.

Type	Number

Required	Yes
Default	0.7 (70%)

threads

Global number of Moobots (threads) per Moolet. Override this setting by using the threads property in individual Moolet configurations.

Type	Integer
Required	Yes
Default	10

Note

Do not change this setting unless instructed by Cisco Support.

Configure the Message Bus

The Moogsoft Messaging System (MooMS) is the Message Bus component of Cisco Crosswork Situation Manager and shares event data. This is subscribed to by the various Moolets.

The Message Bus is a publish-subscribe message brokering system implemented with RabbitMQ which uses AMQP, an open standard for message-orientated middleware, over TCP.

Message Handling

The Message Bus handles the data it receives (e.g. raw event data, new alerts, Situation activity etc) by placing it in queues, which are lines of messages waiting to be handled.

Cisco Crosswork Situation Manager does not enforce any size or time limits on queues, so the maximum number of messages in a queue is limited by the available RAM and disk space on the server. It also depends on the size of the alerts and Situations being generated. The size limit is 128kb for Alerts and 64kb for Situations.

Once the maximum number of messages has been reached, the broker drops messages from the front of the queue to make room for new messages. By default, Cisco Crosswork Situation Manager applications use exclusive transient queues. For example, if Cisco Crosswork Situation Manager or the broker shuts down or dies then the queue and all of its messages are lost. Durable queues can be enabled using the **message_persistence** setting in **$MOOGSOFT_HOME/config/system.conf** (see Message Persistence).

For more information see the RabbitMQ docs on queues and queue length.

Default Configuration

Cisco Crosswork Situation Manager is installed with a single RabbitMQ broker by default, running on the same machine as the other components (LAMs, Moogfarmd, the Moolets, etc).

The out-of-the-box configuration in **$MOOGSOFT_HOME/config/system.conf** is as follows:

```
port: 5672
zone: <none>
username: moogsoft
password: m00gs0ft
```

The username and password must match the Message Bus broker configuration. If commented out, a default "guest" user will be used (guest: guest).

Zones

You can use zones, or virtual hosts, to share a single RabbitMQ broker cluster among multiple instances of Cisco Crosswork Situation Manager.

If multiple instances of Cisco Crosswork Situation Manager share a single RabbitMQ broker then each instance uses a different zone name to prevent message interference. In RabbitMQ, a zone is called a virtual host (**vhost**). Clients connecting to one **vhost** cannot see messages sent to a different **vhost**.

The default deployment does not use zones. If you specify a zone name, you must also configure a **vhost** with the same name in the RabbitMQ broker.

By default, Cisco Crosswork Situation Manager clients connect to the **vhost** specified during **moog-init.sh** setup with the **moogsoft** username and the password.

For distributed installations using multiple RabbitMQ brokers, this must be configured. A zone (**vhost**) name is required by the **moog-init.sh** setup script. See Message System Deployment.

Message Persistence

You can control and minimize message loss during a shutdown or failure using the following settings in **$MOOGSOFT_HOME/config/system.conf**. See the section below for the configurable properties. For other properties, see System ConfigurationSystem Configuration

connections_per_producer_pool

The number of connections to use for each message sender pool. For example, if a message sender pool has 20 channels and this property is set to 2, the channels are split across both connections so that each has 10 channels. To configure this property, you must manually add it to the **mooms** section.

Type	Integer
Required	No
Default	2

zone

Name of the zone.

Type	String
Required	No
Default	N/A

brokers

Hostname and port number of the RabbitMQ broker.

Type	Array
Required	No
Default	**"host" : "localhost", "port" : 5672**

username

Username of the RabbitMQ user. This needs to match the RabbitMQ broker configuration. If commented out, it uses the default "guest" user.

Type	String
Required	No
Default	**guest**

password

Password for the RabbitMQ user. You can choose to either have a password or an encrypted password, you cannot use both.

Type	String
Required	Yes. If you are not using **encrypted password**.
Default	**guest**

encrypted_password

Encrypted password for the RabbitMQ user. You can choose to either have a password or an encrypted password, you cannot use both. See Moog Encryptor if you want to encrypt your password.

Type	String
Required	Yes. If you are not using **password**.
Default	N/A

threads

Number of threads a process can create in order to consume the messages from the Message Bus. If not specified, the thread limit = (Number of processors x 2) + 1. Altering this limit affects the performance of Cisco Crosswork Situation Manager processes such as Moogfarmd and Moogpoller.

If your logs indicate an issue in creating threads, Cisco advises that you increase the ulimit, the maximum number of file descriptors each process can use, for the Cisco Crosswork Situation Manager user. You can set this limit in **/etc/security/limits.conf**.

Type	Integer
Required	No
Default	**10**

message_persistance

Controls whether RabbitMQ persists important messages. Message queues are durable by default and data is replicated between nodes in High Availability mode. Setting this value to **false** means that replicated data is not stored to disk.

Type	Boolean
Required	No
Default	**true**

message_prefetch

Controls how many messages a process can take from the Message Bus and store in memory as a buffer for processing. This configuration allows processes to regulate message consumption which can ease backlog and memory consumption issues. The higher the number, the more messages held in the

process's memory. Set to 0 for unlimited processing. To achieve high availability of messages and ensure messages are processed, the value of this should be higher than 0.

Type	Integer
Required	No
Default	**0**

max_retries

Maximum number of attempts to resend a message that failed to send. Cisco Crosswork Situation Manager only attempts a retry when there is a network outage or if **cache_on_failure** is enabled.

You can use this in conjunction with the **retry_interval** property. For example, a combination of 100 maximum retries and 200 milliseconds for retry interval leads to a total of 20 seconds. The combined default value for these properties was chosen to handle the typical time for a broker failover in a clustered environment.

Type	Integer
Required	No
Default	**100**

retry_interval

Maximum length of time to wait in milliseconds between each attempt to retry and send a message that failed to send.

You can use this in conjunction with the **max_retries** property. The combined value for these properties was chosen to handle the typical time for broker failover in a clustered environment.

Type	Integer
Required	No
Default	**200**

cache_on_failure

Controls whether Cisco Crosswork Situation Manager caches the message internally and resends it if there is an initial retry failure. The system attempts to resend any cached messages in the order they were cached until the time-to-live value, defined by the **cache_ttl** property, is reached.

Type	Boolean
Required	No
Default	**false**

cache_ttl

Length of time in seconds that Cisco Crosswork Situation Manager keeps cached messages in the cache list before discarding them. If a message is not successfully resent within this timeframe it is still discarded.

This defaults to 900 seconds (15 minutes). Increasing this value has a direct impact on sender process memory.

Type	Integer
Required	No

Default	**900**

confirmation_timeout

Length of time in milliseconds to wait for the Message Bus to confirm that a broker has received a message. Cisco does not advise changing this value.

Type	Integer
Required	No
Default	**2000**

Message System Deployment

The Message Bus is the message system for Cisco Crosswork Situation Manager, implemented with RabbitMQ. By default, the Cisco Crosswork Situation Manager installation includes with a single RabbitMQ broker running on the same server as the other Cisco Crosswork Situation Manager components (LAMs, moogfarmd, the Moolets, etc.). You can also configure Cisco Crosswork Situation Manager as a distributed system with multiple RabbitMQ broker hosts.

If you encounter any errors or issues with your deployment see Troubleshooting.

Distributed Deployment

Depending on the systems you monitor, you can increase the performance and reliability of your Cisco Crosswork Situation Manager deployment with distributed RabbitMQ brokers running on different hosts. Execute the following procedure on each RabbitMQ broker host to install a distributed messaging system:

- Install the Erlang package built by RabbitMQ:

```
yum -y install https://github.com/rabbitmq/erlang-
rpm/releases/download/v20.1.7/erlang-20.1.7-1.el7.centos.x86_64.rpm
```

- Set up RabbitMQ yum repository:

```
curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-
server/script.rpm.sh | sudo bash
```

- Install RabbitMQ:

```
yum -y install rabbitmq-server-3.7.4
```

- Copy **rabbitmq.config** from **$MOOGSOFT_HOME/etc/cots/rabbitmq/rabbitmq.config** and add it to the following location on each RabbitMQ broker host:

```
/etc/rabbitmq/rabbitmq.config
```

- Run the following commands:

```
chkconfig rabbitmq-server on
service rabbitmq-server start
```

- Create a new zone (a RabbitMQ "vhost") on each remote RabbitMQ broker and set the permissions for the default user:

```
rabbitmqctl add_vhost <zone>
Creating vhost "<zone>" ...

rabbitmqctl add_user moogsoft <password>
Creating user "moogsoft" ...

rabbitmqctl set_permissions -p <zone> moogsoft ".*" ".*" ".*"
Setting permissions for user "moogsoft" in vhost "<zone>" ...
```

- Edit the "**mooms**" section in **system.conf** on the Cisco host system, to point to the correct IP addresses and ports (two specified in the example below):

```
"mooms" :
   {
       "zone"     : "<zone>",
       "brokers" : [
           {
               "host"     : "172.16.87.131",
               "port"     : 5672
           },
           {

               "host"     : "172.16.87.135",
               "port"     : 5672
           }
       ]
       ,"username"        : "moogsoft",
       "password"        : "<password>"
   },
```

Cluster Message Bus Brokers

The Message Bus broker is a logical grouping containing one or more Erlang nodes each running RabbitMQ and sharing vhosts, users, queues etc. If you have multiple RabbitMQ brokers running, you should cluster them.

For more information about broker clustering, refer to the RabbitMQ documentation.

Enable Queue Mirroring

As part of a Cisco Crosswork Situation Manager High Availability (HA) deployment that employs message persistence, you must set up mirroring for the relevant durable queues across all nodes in a RabbitMQ cluster.

To enable queue mirroring, run the following command from any host running a broker in the RabbitMQ cluster:

```
rabbitmqctl set_policy -p <zone> ha-all ".+\.HA" '{"ha-mode":"all"}'
```

For the **<zone>**, specify the zone you used when you initialized your system. For example, if the zone is set to Cisco Crosswork Situation Manager:

```
rabbitmqctl set_policy -p MoogsoftAIOps ha-all ".+\.HA" '{"ha-mode":"all"}'
```

This command configures mirroring for all the *.HA queues across all RabbitMQ brokers in the cluster.

Run the following command from any host running a broker in the RabbitMQ cluster to verify the policy is enabled:

```
rabbitmqctl -p <zone> list_policies
```

For example:

```
rabbitmqctl -p MoogsoftAIOps list_policies
Listing policies for vhost "MoogsoftAIOps" ...
MoogsoftAIOps ha-all .+\.HA all {"ha-mode":"all"} 0
```

For more information about queue mirroring, refer to the RabbitMQ docs.

Message System Troubleshooting

- RabbitMQ Broker Fails to Start
- Typical Error Messages
- First startup of RabbitMQ broker
- LAMs fail to start from command line
- Manually configure IP address and port
- Manually set up a user or zone (vhost)

The Message Bus (sometimes called MooMs) is the message system for Cisco Crosswork Situation Manager, implemented with RabbitMQ.

This guide outlines some common issues with the Message Bus deployment and offers alternative solutions.

Open the Management Console

You can launch the RabbitMQ management console directly from the Cisco Crosswork Situation Manager UI. This provides useful statistics when debugging.

To open the console, go to Settings > Self Monitoring > Message Bus and click Launch Message Bus Console ... You can log in using the default credentials:

Username: **moogsoft**

Password: **m00gs0ft**

These are defined in **system.conf**. If commented out, a default '**guest**' user can be used.

Examine the Log Files

Troubleshooting your Message Bus deployment often requires examining log files. The default locations of log files are as follows:

- moog_farmd and LAMs - **/usr/log/moogsoft/$SERVICE_NAME.log** where **$SERVICE_NAME** is the process, for example **socketlamd** for the Socket LAM
- Tomcat -**/usr/share/apache-tomcat/logs/catalina.out**
- RabbitMQ - **$RABBITMQ_HOME/var/log/rabbitmq**

RabbitMQ Broker Fails to Start

If RabbitMQ broker fails to start and Security-Enhanced Linux (SELinux) is enabled, it may be related to this. SELinux and similar mechanisms such as firewalls may prevent RabbitMQ from binding to a port and starting up.

If SELinux is enabled, check that the following ports can be opened:

- 4369 (Erlang port mapper daemon)
- 25672 (Erlang distribution)
- 5672, 5671 (AMQP 0-9-1 without and with TLS)
- 15672 (if management plugin is enabled)

You may need to configure RabbitMQ to use different ports.

For more information, refer to the RabbitMQ installation documentation.

If the RabbitMQ broker fails to start it may be due to file permissions, you may see errors such as:

```
{error_logger,{{2015,9,1},{21,26,14}},"Failed to create cookie file
'/home/moogsoft/.erlang.cookie': eacces",[]}

{error_logger,{{2015,9,1},{21,26,14}},crash_report,[[{initial_call,{auth,init,['
Argument__1']}},{pid,<0.21.0>},{registered_name,[]},{error_info,{exit,{"Failed
to create cookie file '/home/moogsoft/.erlang.cookie':
eacces",[{auth,init_cookie,0,[{file,"auth.erl"},{line,286}]},{auth,init,1,[{file
,"auth.erl"},{line,140}]},{gen_server,init_it,6,[{file,"gen_server.erl"},{line,3
28}]},{proc_lib,init_p_do_apply,3,[{file,"proc_lib.erl"},{line,239}]}]},[{gen_se
rver,init_it,6,[{file,"gen_server.erl"},{line,352}]},{proc_lib,init_p_do_apply,3
,[{file,"proc_lib.erl"},{line,239}]}]}},{ancestors,[net_sup,kernel_sup,<0.10.0>]
},{messages,[]},{links,[<0.19.0>]},{dictionary,[]},{trap_exit,true},{status,runn
ing},{heap_size,610},{stack_size,27},{reductions,975}],[]]}...
```

When the RabbitMQ broker is running as a service, use the following command to check that it is running:

```
service rabbitmq-server status
```

Typical Error Messages

The section below will outline examples and solutions to typical error messages with RabbitMQ.

Connection refused/ Unable to create RabbitMQ connection

If the RabbitMQ broker appears to be down or unreachable, trying to start an Cisco Crosswork Situation Manager component gives warnings such as:

```
WARN : [main ][20150812 16:09:54.792 +0100] [CMoomsFactory.java]:707 +|Unable to
create RabbitMQ connection : [amqp://localhost:5672/ZONE]|+
WARN : [main ][20150812 16:09:54.792 +0100] [CMoomsFactory.java]:256 +|Unable to
create RabbitMQ connection : [java.net.ConnectException: Connection refused]|+
WARN : [main ][20150812 16:09:54.793 +0100] [CMoomsFactory.java]:707 +|Unable to
create RabbitMQ connection : [amqp://localhost:5672/ZONE]|+
WARN : [main ][20150812 16:09:54.793 +0100] [CJNIMoomsWrapper.java]:253
+|Problem during mooms setup, retrying|+
```

The structure of the amqp url is: **`amqp://<hostname>:<port>/<zone>`**

Solution:

Check if the RabbitMQ broker is running:

```
service rabbitmq-server status
```

If it isn't running, see RabbitMQ broker fails to start (above).

If it is running, check the Message Bus configuration in the 'mooms' section of **system.conf.**

AlreadyClosedException/broker forced connection closure

If Cisco Crosswork Situation Manager components are running, the RabbitMQ broker going down or becoming unreachable gives warnings such:

```
WARN : [Thread-][20150812 16:15:42.295 +0100] [CLogger.java]:337 +|Problem
sending message id : [4115e512-aa63-44d5-bdc9-8ed164cd75e5]
com.rabbitmq.client.AlreadyClosedException: connection is already closed due to
connection error; protocol method: #method<connection.close>(reply-code=320,
reply-text=CONNECTION_FORCED - broker forced connection closure with reason
'shutdown', class-id=0, method-id=0)
at com.rabbitmq.client.impl.AMQChannel.ensureIsOpen(AMQChannel.java:195)
at com.rabbitmq.client.impl.AMQChannel.transmit(AMQChannel.java:309)
at com.rabbitmq.client.impl.ChannelN.basicPublish(ChannelN.java:657)
at com.rabbitmq.client.impl.ChannelN.basicPublish(ChannelN.java:640)
at com.rabbitmq.client.impl.ChannelN.basicPublish(ChannelN.java:631)
at
com.rabbitmq.client.impl.recovery.AutorecoveringChannel.basicPublish(Autorecover
ingChannel.java:168)
at com.moogsoft.mooms.CMoomsMessageSender.send(CMoomsMessageSender.java:530)
at com.moogsoft.mooms.CMoomsMessageSender.send(CMoomsMessageSender.java:448)
at com.moogsoft.mooms.CMoomsMessageSender.send(CMoomsMessageSender.java:264)
at
com.moogsoft.mooms.CMoomsMessageSenderPool.send(CMoomsMessageSenderPool.java:378
)
at com.moogsoft.mooms.CJNIMoomsWrapper.sendEvent(CJNIMoomsWrapper.java:288)
|+
```

Note

If Cisco Crosswork Situation Manager is configured to connect to RabbitMQ brokers in a high availability environment, during a RabbitMQ broker fail-over, warning messages may be logged for a short time

Solution:

If the problem is not temporary, check if the RabbitMQ broker is running:

```
service rabbitmq-server status
```

Problem during mooms setup, retrying

A Cisco Crosswork Situation Manager component trying to connect to a non-existent zone (vhost) in a RabbitMQ broker gives warnings such as:

```
DEBUG: [main ][20150812 16:24:00.764 +0100] [CMoomsFactory.java]:206 +|Setting
factory zone to : [fish]|+
WARN : [main ][20150812 16:24:03.825 +0100] [CMoomsFactory.java]:707 +|Unable to
create RabbitMQ connection : [amqp://localhost:5672/fish]|+
WARN : [main ][20150812 16:24:03.825 +0100] [CMoomsFactory.java]:256 +|Unable to
create RabbitMQ connection : [java.io.IOException]|+
WARN : [main ][20150812 16:24:06.373 +0100] [CMoomsFactory.java]:707 +|Unable to
create RabbitMQ connection : [amqp://localhost:5672/fish]|+
WARN : [main ][20150812 16:24:06.373 +0100] [CJNIMoomsWrapper.java]:253
+|Problem during mooms setup, retrying|+
```

Solution:

Check you have the correct zone configured in the 'mooms' section of **system.conf** and that the zone (vhost) has been created in the RabbitMQ broker.

The best way to check that the zone (vhost) has been created in the RabbitMQ broker is to use the RabbitMQ management console.

If the zone (vhost) has not been created, manually create it via the command line, or via the RabbitMQ Management console.

Once the zone has been created, the user defined in **system.conf** must be given permissions to access the new zone.

AuthenticationFailureException: ACCESS_REFUSED

A Cisco Crosswork Situation Manager component trying to connect to a valid zone (vhost) in a RabbitMQ broker, but with wrong authentication details gives warnings such as:

```
WARN : [main ][20150812 16:20:29.760 +0100] [CMoomsFactory.java]:707 +|Unable to
create RabbitMQ connection : [amqp://jimmy@localhost:5672/null]|+
WARN : [main ][20150812 16:20:29.760 +0100] [CMoomsFactory.java]:256 +|Unable to
create RabbitMQ connection :
[com.rabbitmq.client.AuthenticationFailureException: ACCESS_REFUSED - Login was
refused using authentication mechanism PLAIN. For details see the broker
logfile.]|+
WARN : [main ][20150812 16:20:29.805 +0100] [CMoomsFactory.java]:707 +|Unable to
create RabbitMQ connection : [amqp://jimmy@localhost:5672/null]|+
WARN : [main ][20150812 16:20:29.805 +0100] [CJNIMoomsWrapper.java]:253
+|Problem during mooms setup, retrying|+
```

Solution:

Check you have the correct username and password in the 'mooms' section of **system.conf** and that they match those defined in the RabbitMQ broker. If they are correct in **system.conf** then you must correct it in the RabbitMQ broker. Do this either via the command line, or via the RabbitMQ management console.

Also see RabbitMQ broker fails to start (above).

First startup of RabbitMQ broker

The first time a RabbitMQ broker is started, it creates an 'account' with the default user and password from **rabbitmq.config**.

If this information is subsequently edited in **rabbitmq.config**, and the RabbitMQ broker is restarted, the 'account' is not created, which can be confusing.

If the RabbitMQ broker has been started before, then the 'account' will need to be added manually (see Manually set up a user) rather than by defining a default user in the **rabbitmq.config** file.

LAMs fail to start from command line

If LAMs run from the command line or as a service result in the following error:

```
[root@moogbox2 bin]# ./socket_lam
./socket_lam: error while loading shared libraries: libjvm.so: cannot open
shared object file: No such file or directory
```

...it may be because **/usr/java/jdk1.8.0_20/jre/lib/amd64/server** has not been added to the **LD_LIBRARY_PATH**.

- To run the LAMs via a command line, a change the **LD_LIBRARY_PATH** to be as follows (the default **initd** files have this modification made):

```
export
LD_LIBRARY_PATH=$MOOGSOFT_HOME/lib:/usr/GNUstep/Local/Library/Libraries:/usr/GNU
step/System/Library/Libraries:$JAVA_HOME/jre/lib/amd64/server
```

Manually configure IP address and port

In some environments (such as SELinux) you may need to configure a RabbitMQ broker to listen on a different IP address and port.

To do this:

- Configure the contents of **/etc/rabbitmq/rabbitmq-env.conf**, for example:

```
RABBITMQ_NODE_IP_ADDRESS="172.168.87.131
RABBIT_NODE_PORT="5678"
```

- Restart the rabbitmq-server service:

```
service rabbitmq-server restart
```

Manually set up a user or zone (vhost)

To help troubleshoot an existing RabbitMQ broker, you may want to manually set up a user or zone (vhost).

- To manually set up a new user in the RabbitMQ broker, run the following command (using your own user, password and zone):

```
rabbitmqctl add_user <user> <password>
rabbitmqctl set_permissions -p <zone> <user> ".*" ".*" ".*"
```

Also ensure the username, password and zone in the 'mooms' section of **system.conf** match those defined in the RabbitMQ broker with the above commands.

- To manually set up a new zone in the RabbitMQ broker, run the following command (using your own user and zone):

```
rabbitmqctl add_vhost <zone>
rabbitmqctl set_permissions -p <zone> <user> ".*" ".*" ".*"
```

Also ensure the username and zone in the 'mooms' section of **system.conf** match those defined in the RabbitMQ broker with the above commands.

Message System SSL

The Message Bus system (MooMs) can be configured to operate using SSL connections to provide secure and authorized connectivity.

The message system for Cisco Crosswork Situation Manager is implemented with RabbitMQ. By default, Cisco Crosswork Situation Manager provides rabbitmq.config which does not start RabbitMQ in SSL mode.

To enable RabbitMQ to run in SSL mode, see the Rabbit MQ documentation.

Configure Cisco Crosswork Situation Manager to use SSL with the Message Bus

Once RabbitMQ has been configured to use SSL, Cisco Crosswork Situation Manager needs to be configured to use the RabbitMQ broker's SSL port, as well as the SSL certificates and keys to enable secure and authorized connection to these brokers if required by the SSL configuration set on RabbitMQ.

Below is an example of full SSL Message Bus configuration in system.conf:

system.conf

```
###############################################################################
# SSL configuration can be used to provide a means of secure                  #
# communication between a Moog process and MooMS. MooMS can be setup          #
# with options to accept SSL connections with or without providing            #
# the relevant certificates and keys.                                         #
# Three modes of SSL are available:                                           #
# 1. No SSL      - SSL configuration is not specified                         #
# 2. Express SSL - This is where SSL configuration is specified, but          #
#                  empty or only the SSL protocol is set and specific         #
#                  certificates do not need to specified.                     #
# 3. Custom SSL  - This is where all the SSL configuration and                #
#                  certificates needed are specified to enable secure         #
#                  and authorised communication to MooMS.                     #
#                  Note that Client key and certificate are optional.         #
#                  If neither of those are specified, then client             #
#                  certification verification will not be performed.          #
###############################################################################

"ssl" :
{
    # Specify the SSL Protocol to use.
    # If the configuration is not specified, "TLSv1.2" will be used
    # by default.
    # JRE 8 supports "TLSv1.2", "TLSv1.1", "TLSv1", "SSLv3"
    #
    "ssl_protocol" : "TLSv1.2",
    #
    # The location of the SSL certificate, key files.
    #
    # Relative pathing can be used, i.e. '.' to mean current directory,
    # '../server.pem' or '../../server.pem' etc. If neither relative
    # nor absolute (using '/') path is used then $MOOGSOFT_HOME is
    # prepended to it.
    # i.e. "config/server.pem" becomes "$MOOGSOFT_HOME/config/server.pem"
    #
    # Specify the server certificate.
    #
    "server_cert_file" : "server.pem",
    #
    # Enable client authentication by specifying the client certificate
    # and key files below.
    # The key file has to be in PKCS#8 format.
    #
    "client_cert_file" : "client.pem",
    "client_key_file"  : "client.key"
}
```

Express SSL

Cisco Crosswork Situation Manager can be configured to connect to the RabbitMQ server without validating any certificates or attempting to authorize the client.

If the RabbitMQ server has been configured to reject clients that do not present valid certificates then this SSL mode will not work, Cisco Crosswork Situation Manager will need to be configured with the correct certificates and keys to establish connectivity. To enable express SSL mode simply uncomment "ssl" configuration block, optionally specify the "ssl_protocol" configuration:

Express SSL

```
"ssl" :
{
    # Specify the SSL Protocol to use.
    # If the configuration is not specified, "TLSv1.2" will be used
    # by default.
    # JDK 8 supports "TLSv1.2", "TLSv1.1", "TLSv1", "SSLv3"
    #
    "ssl_protocol" : "TLSv1.2"
}
```

Custom SSL

Cisco Crosswork Situation Manager can be configured to connect to the RabbitMQ server using a specific server certificate, and if RabbitMQ has been enabled with Client Authentication then Cisco Crosswork Situation Manager can be configured with the client key and client certificate to authenticate with RabbitMQ.

Client Authentication is optional functionality, to run Cisco Crosswork Situation Manager with just a specific server certificate simply comment out the client_cert_file and client_key_file entries.

Note

If Client Authentication is used, the "client_key_file" must be in a PKCS#8 Format. The following command can be run to convert a private key in to PKCS#8 format:

openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in key.pem -out client.key

An example of Cisco Crosswork Situation Manager specifying full SSL configuration, connecting to a RabbitMQ which requires Client Authentication. The example also shows how you can organise the server and client SSL files in sub-folders:

Custom SSL

```
"ssl" :
{
    # Specify the SSL Protocol to use.
    # If the configuration is not specified, "TLSv1.2" will be used
    # by default.
    # JRE 8 supports "TLSv1.2", "TLSv1.1", "TLSv1", "SSLv3"
    #
    "ssl_protocol" : "TLSv1.2",
    #
    # The location of the SSL certificate, key files.
    #
    # Relative pathing can be used, i.e. '.' to mean current directory,
    # '../server.pem' or '../../server.pem' etc. If neither relative
```

```
    # nor absolute (using '/') path is used then $MOOGSOFT_HOME is
    # prepended to it.
    # i.e. "config/server.pem" becomes "$MOOGSOFT_HOME/config/server.pem"
    #
    # Specify the server certificate.
    #
    "server_cert_file" : "server/server.pem",
    #
    # Enable client authentication by specifying the client certificate
    # and key files below.
    # The key file has to be in PKCS#8 format.
    #
    "client_cert_file" : "client/client.pem",
    "client_key_file"  : "client/client.key"
}
```

Note

To disable SSL connectivity with the Message Bus, change the port number for the brokers back to the non-SSL port (typically 5672) and comment out the "ssl" section in system.conf.

Configure Search and Indexing

Cisco Crosswork Situation Manager uses Elasticsearch to provide search and data indexing functions.

You can control the Elasticsearch service using the following service script:

```
/etc/init.d/elasticsearch [start|restart|stop]
```

All Elasticsearch logs are stored in following location:

```
/var/log/elasticsearch/
```

Index Alerts and Situations

Two tools are used to index alerts and Situations: the Indexer Moolet and the Moog Indexer utility.

Indexer Moolet

The Indexer listens for new alerts and Situations on the Message Bus and indexes them. Cisco Crosswork Situation Manager indexes alerts and Situations as soon as they are are created or modified so that they are immediately searchable.

You can configure the Indexer in **`$MOOGSOFT_HOME/config/moolets/indexer.conf`** using the following parameters:

enable_private_teams

Set to true if you limit team permissions based upon services, Situations, or alerts assigned to the team. The the indexer applies team permissions to the indexes.Manage Teams

If disabled, the Indexer will index all alerts and Situations present in Cisco Crosswork Situation Manager.

Type: Boolean

Default: **`False`**

full_scan_batch_size

The maximum number of alerts or Situations the Indexer scans in each batch. This is useful because it is not possible to load all alerts to the memory at once.

By default the Indexer scans through batches of one thousand alerts or Situations.

Type: Integer

Default: **1000**

full_scan_wait

The number of seconds the Indexer waits between batches. This frees up the CPU and memory used to index each batch.

It is set to zero by default so the Indexer will not wait between batches.

Type: Integer

Default: **0**

full_scan_at

Determines the exact time when Indexer runs a full scan. This allows you to ensure the accuracy of search data once per day by performing a full reindex. If left empty, the Indexer does not perform a full scan.

Type: Time (HH:mm:ss)

Default: **"02:12:35"**

full_scan_at_startup

If enabled, the Indexer performs a full scan when it starts. This is useful if you are not using the scheduled scan and only restart Moogfarmd once a week.

Type: Boolean

Default: **false**

historic_scan_frequency

Determines how frequently the Indexer performs a full scan of both active and historic databases. By default, the Indexer scans both databases every three days.

Type: Integer

Default: **3**

By default the Indexer is configured as follows:

```
# Set to false to disable private teams indexing.
enable_private_teams: false,

# Maximal full scan batch size
full_scan_batch_size: 1000,

# How many seconds to wait between batches (0 not to wait)
full_scan_wait: 0,

# When to run the full scan (HH:mm:ss) leave empty to disable full scan
(HH:mm:ss)
```

```
full_scan_at: "02:12:35",

# Do we want to run full scan when the moolet starts?
full_scan_at_startup: false

# Scan the historic data once every how many full scans
historic_scan_frequency: 3
```

Moog Indexer

Before you can run the indexer utility, you must start Moogfarmd with a running Indexer Moolet. The **moog_indexer** accepts the following options:

Argument	Input	Description
-h,--help	-	Displays the help text with arguments that can be used with the utility.
-f, --full	-	Scans both the active and historic data. Use this argument if you want data from both databases to be indexed.
-i,--in <arg>	Integer	Schedule full index to run in a set amount of time (in hours). This can be a decimal. For example, 0.1 = 6 minutes.
-l,--loglevel <arg>	WARN|INFO|DEBUG|TRACE	Specify the log level to choose the amount of debug output. Defaults to INFO.
-n,--now	-	Schedules a full index to run immediately.
-r,--report	-	Request report from on the last performed full scan index. This report will show the status of previous runs within the lifetime of the moogfarmd process and any runs still in progress. If moogfarmd is restarted, the `-r` argument will not return any data.

Note

If you use Private Teams mode, meaning one or more Roles do NOT have the all_data permission set, then you must run both the initial 'full index' and the 'incremental index crontab' moog_indexer commands with the `-p` argument. If not, users in one Team will be able to see search results for other Teams.

Tune your MySQL database to ensure indexing runs as quickly as possible. See either the Percona or MySQL websites for information on tuning and optimization.

An output example is shown below:

```
[root@myhost home]# moog_indexer -r
Got report:
        05/10/17 13:43:06 - Starting full scan
        05/10/17 13:43:06 - Scanning for alerts
        05/10/17 13:43:07 - Scanned: [177] alerts
        05/10/17 13:43:07 - Scanning for situations
        05/10/17 13:43:07 - Scanned: [44] situations
        05/10/17 13:43:07 - Full scan complete
        05/10/17 13:43:22 - Starting full scan
        05/10/17 13:43:22 - Scanning for alerts
        05/10/17 13:43:22 - Scanned: [204] alerts
        05/10/17 13:43:22 - Scanning for situations
        05/10/17 13:43:23 - Scanned: [55] situations
        05/10/17 13:43:23 - Full scan complete
```

Warning

Before you upgrade to Cisco Crosswork Situation Manager V6.2.1 or later, remove or disable the crontab jobs for the old indexer utility.

Elasticsearch Details

Elasticsearch runs on port 9200 by default.

To make Elasticsearch available externally and listen on the external host IP address, run the following command:

```
$MOOGSOFT_HOME/bin/utils/moog_init_search.sh -r
```

The script updates the Elasticsearch configuration and restarts the service.

Log Levels Reference

Cisco Crosswork Situation Manager components generate log files to report their activity. Log messages sent from these components use the following log levels:Configure Logging

Level	Description
ERROR	Errors have occurred but the application is still able to run.
WARN	Indicates something potentially serious or harmful has happened to your application.
INFO	Informational messages that report the application's normal behaviour.
DEBUG	Diagnostic and granular information that can be useful for debugging an application.
TRACE	Similar to DEBUG but even more fine-grained information.

Configure Labs Features

Cisco Crosswork Situation Manager Labs offers a preview of unreleased features. Go to Settings > System > Labs to view available Labs features for the current release.

There are no unreleased features in v8.0.

Enable Situation Room Plugins

You can add configurable third party plugin tabs to the Situation Room in Cisco Crosswork Situation Manager that relate to the Situation. For example, you can link to a ServiceNow incident that is mapped to the Situation in question.

Cisco Crosswork Situation Manager requires a **`link_definition`** and **`custom_info`** column to enable the Situation Room Plugin.

Note

The Show ServiceNow Incident in the Situation Room check box in System Administration, Integrations, ServiceNow adds the plugin automatically

Two use cases are:

- Conditional, based on contents of a field being present in the Situation (if the field is left empty, the plugin is disabled).
- Universal for all Situations.

Further to these use cases, the **link_definition** can also have the same use cases:

- Conditional, taking attributes of the Situation and passing to the linked application.
- Generic, passing no details about the Situation.

Implementation

The moogdb.sitroom_plugins table has the following columns:

- Title.
- Its associated Situation field (from the moogdb.sigs table and can be custom_info.<field>).
- The link definition to use.

```
mysql> describe moogdb.sitroom_plugins;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| id            | int(11)      | NO   | PRI | NULL    | auto_increment |
| title         | varchar(32)  | YES  | UNI | NULL    |                |
| internal_name | varchar(255) | YES  |     | NULL    |                |
| link_name     | varchar(32)  | YES  | MUL | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

Examples

Assuming Situation 14 had already been linked with ServiceNow Incident INC0000055 using the following SQL:

```
update moogdb.sigs set custom_info='{"servicenow_id":"INC0000055"}' where
sig_id=14;
```

Enable a ServiceNow tab in the Situation Room

1. Define a link_definition to point to the ServiceNow URL, and use the **$value** dynamic placeholder to be replaced with the incident number when launched.

```
insert into moogdb.link_definitions (name, link) values ('servicenow',
'https://<your-server-here>/incident.do?sysparm_query=number%3D$value');
```

2. Add an entry into the sitroom_plugins table to define:

- The tab.
- The Situation field to be used for **$value**.
- The **link_definition**.

```
insert into moogdb.sitroom_plugins (title, internal_name, link_name) values
('ServiceNow', 'custom_info.servicenow_id', 'servicenow');
```

If desired, you can define multiple plugin tabs (with unique title) using the same or different Situation fields.

Additional configuration

You must configure your browser to allow third party cookies from these URLs, otherwise, remote websites may not display within the tab area. To do this, either enable third-party cookies globally, or, allow the specific URLs to set third party cookies as exceptions.

You must enable iFrames in ServiceNow to use Situation Room Plugins. They are disabled by default. For more information refer to the ServiceNow documentation, for example High Security Settings. You may also need to disable "same origin" settings.

ServiceNow integration

Configure ServiceNow using the UI. See the Integrate ServiceNow section in ServiceNow for details.ServiceNow

MoogDb V2

You can query and manipulate a variety of entities in the Cisco Crosswork Situation Manager database using the MoogDb v2 Moobot module.

The module uses various methods to retrieve information from MoogDb and update components of Cisco Crosswork Situation Manager including alerts, Situations, users and teams.

All MoogDb v2 methods that update the database also publish information about the appropriate updated entities on the Configure the Message Bus, so any updated information automatically appears in Cisco Crosswork Situation Manager when the relevant method is called.

Load MoogDb v2

You can load the MoogDb v2 module into any standard MooBot by defining a new global object called **moogdb** at the top of the JavaScript file:

```
var moogdb = MooBot.loadModule('MoogDb.V2');
```

Methods

See MoogDb V2 API Method Reference for details of all the MoogDb v2 API methods.

MoogDb V2 API Method Reference

This is a reference list for the MoogDb v2 API methods. Follow the links to see the details and examples of each method.

Alerts

The following methods relate to alerts:

- addAlertToSituation: Adds an alert to a Situation.
- assignAlert: Assigns a user as the owner of an alert.
- assignAndAcknowledgeAlert: Assigns and acknowledges the specified user as the owner of a specified alert.

- closeAlert: Closes one or more alerts.
- createAlert: Creates or updates an alert.
- deassignAlert: Removes the assignment of the current owner from an alert.
- getAlert: Returns details of an alert.
- getAlertActions: Returns the actions for one or more alerts or for a time period.
- getAlertCustomInfo: Returns custom information for an alert.
- getAlertIds: Returns the total number of alerts, and a list of the alert IDs, for an alert filter and a limit.
- reload: Takes a Situation or alert type of CEvent and refreshes the data in the payload but preserves the metadata.
- removeAlertFromSituation: Removes an alert from a Situation.
- setAlertCustomInfo: Updates the custom information for an alert.
- setAlertSeverity: Sets the severity level for an alert.
- updateAlert: Updates an alert object and uses it to update the database and the Message Bus.
- updateClosedAlert: Updates the description and custom info of a closed alert during the grace period.
- updateCustomInfo: Updates the custom info for an alert or a Situation.

Processes and Maintenance

The following methods relate to the processes and maintenance:

- addProcess: Adds a new process to the database.
- addService: Adds a new external service to the database.
- createMaintenanceWindow: Creates a maintenance window.
- deleteMaintenanceWindow: Deletes a single maintenance window.
- deleteMaintenanceWindows: Deletes maintenance windows that match a filter.
- findMaintenanceWindows: Finds maintenance windows based on a filter and a limit.
- getMaintenanceWindows: Returns maintenance windows based on a start position and a limit.
- getProcesses: Returns a list of processes from the database.
- getQueueName: Returns the queue name for a queue ID.
- getServices: Returns a list of services from the database.
- getToolShares: Returns the shared access for a tool.
- shareToolAccess: Shares access to a tool with other users, teams, or roles, or makes it global so that all users can access it.
- updateMaintenanceWindow: Updates an existing maintenance window.

Situations

The following methods relate to Situations:

- addSigCorrelationInfo: Adds correlation information to a Situation.
- addCorrelationInfo: This method has been superseded; use addSigCorrelationInfo instead.
- addThreadEntry: Adds a new entry to an existing thread in a Situation.
- assignAndAcknowledgeSituation: Assigns and acknowledges a user as the owner of a Situation.
- assignModerator: Assigns a user as the owner of a Situation.
- assignTeamsToSituation: Assigns one or more teams to a Situation.
- checkSituationFlag: Checks whether a flag is associated with a Situation.
- closeSituation: Closes a Situation.
- createThread: Creates a new thread for a Situation.
- createThreadEntry: This method has been superseded; use addThreadEntry instead.
- getActiveSituationIds: Returns the total number of active Situations, and a list of their Situation IDs.
- getPrcLabels: Returns Probable Root Cause (PRC) information for all alerts or specified alerts within a Situation.
- getResolvingThreadEntries: Returns thread entries for a Situation that have been marked as resolving steps.
- getSigCorrelationInfo: Returns all correlation information related to a Situation.
- getSigCustomInfo: Returns all custom info related to a Situation.
- getSituation: Returns details of a Situation.
- getSituationActions: Returns the actions for one or more Situations or for a time period.
- getSituationAlertIds: Returns the total number of alerts, and a list of their alert IDs, for a Situation.
- getSituationFlags: Returns the flags for one or more Situations.
- getSituationHosts: Returns a list of host names for a Situation, either for all alerts in the Situation or just for unique alerts.
- getSituationIds: Returns the total number of Situations, and a list of their Situation IDs, for a Situation filter and a limit.
- getSituationPrimaryTeam: Returns the primary team assigned to a Situation.
- getSituationProcesses: Returns a list of process names for a Situation, and the primary process name, if defined.
- getSituationServices: Returns a list of service names for a Situation, and the primary service name, if defined.

- getSituationsWithFlag: Returns all the Situations which have a specified flag.
- getSituationTopology: Returns the topology of all alerts connected to a Situation.
- getSituationVisualization: Returns the Visualize tab information for a Situation.
- getThreadEntries: Returns thread entries for a thread in a Situation.
- getThreadEntry: Returns a thread entry specified using thread entry ID.
- getTopPrcDetails: Returns the top most likely causal alerts, based on their Probable Root Cause value, for a Situation.
- mergeSituations: Merges two or more Situations.
- moveSituationToCategory: Moves a Situation into a new category.
- moveSituationToQueue: Assigns a Situation to a queue and writes a thread entry if required.
- rateSituation: Applies a rating to a Situation.
- reload: Takes a Situation or alert type of CEvent and refreshes the data in the payload but preserves the metadata.
- removeSigCorrelationInfo: Removes all correlation information related to a Situation.
- removeSituationPrimaryTeam: Removes the primary team from a Situation.
- resolveSituation: Resolves a Situation that is currently open.
- reviveSituation: Revives (sets to Open) a Situation that is currently set to Resolved.
- setPrcLabels: Sets the Probable Root Cause (PRC) labels for alerts within a Situation.
- setResolvingThreadEntry: Sets or clears a thread entry in a Situation as a resolving step.
- setSigCustomInfo: Updates the custom information for a Situation.
- setSituationFlags: Updates the flags associated with a Situation.
- setSituationPrimaryTeam: Sets one of the teams already assigned to a Situation as the primary team.
- setSituationProcesses: Applies a list of processes to a specified Situation.
- setSituationServices: Applies a list of external services to a specified Situation.
- updateClosedSituation: Updates the description and custom info of a closed Situation during the grace period.
- updateCustomInfo: Updates the custom info for an alert or a Situation.
- updateSituation: Updates a Situation object and uses it to update the database and the Message Bus.

User Management

The following methods relate to the management of users, teams and roles:

- createTeam: Creates a new team, by passing an object containing team information.
- createUser: Creates a user, by passing an object containing user properties.

- deleteTeam: Deletes a single team.
- getAllSessionInfo: Returns session information for all users over a period of time.
- getTeam: Returns a team's details by team ID or team name.
- getTeams: Returns details of all teams.
- getTeamsForService: Returns all teams related to a service.
- getTeamSituationIds: Returns the total number of Situations, and a list of the Situation IDs, associated with a team.
- getUser: Returns details of a user.
- getUserName: Returns the username for a user ID.
- getUserRoles: Returns the roles of a user.
- getUsers: Returns details of all users.
- getUserSessionInfo: Returns session information for a single user over a period of time.
- getUserTeams: Returns the team IDs and team names for a user.
- updateTeam: Updates an existing team.
- updateUser: Updates an existing user.

Workflows

The following methods relate to the Workflow Engine:

- createWorkflow: Creates a new workflow at the end of a Workflow Engine Moolet sequence.
- deleteWorkflow: Deletes a workflow from a Workflow Engine Moolet.
- getWorkflowEngineMoolets: Returns a list of Workflow Engine Moolets and the functions available in each.
- getWorkflows: Returns the workflows for a Workflow Engine Moolet.
- reorderWorkflows: Reorders the sequence of workflows within a Workflow Engine Moolet.
- sendToWorkflow: Sends a Moolet Inform message to a workflow in an Inform Workflow Engine.
- updateWorkflow: Updates one or more existing workflows in the Workflow Engine.

addAlertToSituation

A MoogDb v2 method that adds an alert to a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`addAlertToSituation`** takes the following request arguments:

Name	Type	Required	Description

alertId	Number	Yes	Alert ID.
situationId	Number	Yes	Situation ID.

Response

Method **addAlertToSituation** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

addCorrelationInfo

Note

This method has been superseded. Use addSigCorrelationInfo instead. The **addCorrelationInfo** method has been retained for backwards compatibility.

A MoogDb v2 method that adds correlation information (external service name and external entity ID) to a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **addSigCorrelationInfo** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	Situation ID.
service	String		Name of the external service, such as **ServiceNow**.
externalId	String		Identifier that the entity has in the external service, which corresponds to the Situation.

Response

Method **addCorrelationInfo** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

Examples

The following examples demonstrate typical use of method **addCorrelationInfo**:

Response example

A successful request returns **`true`**.

addProcess

A MoogDb v2 method that adds a new process to the database. Processes are external business entities related to business activities that are affected by the incidents that Cisco Crosswork Situation Manager captures in Situations.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`addProcess`** takes the following request arguments:

Name	Type	Required	Description
`process`	String	Yes	Process name.
`description`	String	Yes	Process description.

Response

Method **`addProcess`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

Examples

The following examples demonstrate typical use of method **`addProcess`**:

Response example

A successful request returns **`true`**.

addService

A MoogDb v2 method that adds a new external service to the database. An external service is a business entity monitored by Cisco Crosswork Situation Manager via event streams.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`addService`** takes the following request arguments:

Name	Type	Required	Description
`service`	String	Yes	Name of the external service you are adding.
`description`	String	No	Service description.

Response

Method **`addService`** returns the following response:

Type	Description

Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

Examples

The following examples demonstrate typical use of method **`addService`**:

Response example

A successful request returns **`true`**.

addSigCorrelationInfo

A MoogDb v2 method that adds correlation information (external service name and external entity ID) to a Situation.

This is the recommended method for adding correlation information to a Situation, the **`addCorrelationInfo`** method has been retained for backwards compatibility.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`addSigCorrelationInfo`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.
`service`	String		Name of the external service, such as **`ServiceNow`**.
`externalId`	String		Identifier that the entity has in the external service, which corresponds to the Situation.

Response

Method **`addSigCorrelationInfo`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`addSigCorrelationInfo`**:

Response example

A successful request returns **`true`**.

addThreadEntry

A MoogDb v2 method that adds a new entry to an existing thread in a Situation. Threads are comments or 'story activity' on Situations.

Optionally, you can specify the new entry as being a resolving step.

This method returns the entry ID of the newly created thread entry.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`addThreadEntry`** takes the following request arguments:

Name	Type	Required	Description
`entry`	String	Yes	Description of the new entry you want to add to the existing thread. For example, "And another thing...". Description of the new entry you want to create in the thread. For example, "And another thing...". HTML and XML tags are stripped from the thread entry text. Reserved characters are converted to HTML entities, for example, **`&`** is converted to **`&`**.
`thread_name`	String	Yes	Name of the existing thread.
`user_id`	Number	Yes	A valid user ID.
`sitn_id`	Number	Yes	Situation ID.
`resolving_step`	Boolean	No	Whether or not the thread entry you are adding is a resolving step. Default is **`false`** if not specified,

Response

Method **`addThreadEntry`** returns the following response:

Type	Description
Number	ID of the new thread entry.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **addThreadEntry**:

Request examples

Example request to add an entry "New Entry" to thread "Support" in Situation 158 using user ID 47. The resolving step parameter defaults to **false**.

```
var newThreadEntryID = moogdb.addThreadEntry("New Entry", "Support", 47, 158);
```

Example request to add an entry "New Entry" to thread "Support" in Situation 58 using user ID 47. This thread entry is a resolving step:

```
var newThreadEntryID = moogdb.addThreadEntry("New Entry", "Support", 47, 158,
true)
```

Response example

Example response returning the new thread entry ID:

345

assignAlert

A MoogDb v2 method that assigns a user as the owner of an alert.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **assignAlert** takes the following request arguments:

Name	Type	Required	Description
alertId	Number	Yes	Alert ID.
userId	Number	No, if you specify **username**.	ID of the user to be assigned as the owner of the alert. You must provide the **userId** or the **username**.
username	String	No, if you specify **userId**.	Username of the user to be assigned to the alert. You must provide the **userId** or the **username**.

Response

Method **assignAlert** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

assignAndAcknowledgeAlert

A MoogDb v2 method that assigns and acknowledges a user as the owner of an alert.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`assignAndAcknowledgeAlert`** takes the following request arguments:

Name	Type	Required	Description
`alertId`	Number	Yes	Alert ID.
`userId`	Number	No, if you specify **`username`**.	ID of the user to be assigned as the owner of the alert. You must provide the **`userId`** or the **`username`**.
`username`	String	No, if you specify **`userId`**.	Username of the user to be assigned to the alert. You must provide the **`userId`** or the **`username`**.

Response

Method **`assignAndAcknowledgeAlert`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

assignAndAcknowledgeSituation

A MoogDb v2 method that assigns and acknowledges a user as the owner of a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`assignAndAcknowledgeSituation`** takes the following request arguments:

Name	Type	Required	Description
`SituationId`	Number	Yes	Situation ID.
`userId`	Number	No, if you specify **`username`**.	ID of the user to be assigned as the owner of the Situation. You must provide the **`userId`** or the **`username`**.

username	String	No, if you specify **userId**.	Username of the user to be assigned as the owner of the Situation. You must provide the **userId** or the **username**.

Response

Method **assignAndAcknowledgeSituation** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **assignAndAcknowledgeSituation**:

Response example

A successful request returns **true**.

assignModerator

A MoogDb v2 method that assigns the specified user as the owner of the specified Situation ID.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **assignModerator** takes the following request arguments:

Name	Type	Required	Description
SituationId	Number	Yes	Situation ID.
moderatorId	Number	No, if you specify **username**.	ID of the user to be assigned as the owner of the alert. You must provide the **moderatorId** or the **username**.
username	String	No, if you specify **moderatorId**.	Username of the user to be assigned to the alert. You must provide the **moderatorId** or the **username**.

Response

Method **assignModerator** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`assignModerator`**:

Response example

A successful request returns **`true`**.

assignTeamsToSituation

A MoogDb v2 method that assigns one or more teams to a Situation. Once successfully run, Cisco Crosswork Situation Manager marks the Situation as overridden and the Teams Manager Moolet can no longer modify its team assignment. See Teams Manager Moolet for more information.

The method replaces any teams previously assigned to the Situation. You can also use it to deassign all teams from a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`assignTeamsToSituation`** takes the following request arguments:

Name	Type	Required	Description
`sitn_id`	Number	Yes	Situation ID.
`team_ids`	JSON List	No, if you specify **`team_names`**.	A list of team IDs to assign to the Situation. Specify an empty list to deassign all teams from the Situation.
`team_names`	JSON List	No, if you specify **`team_ids`**.	A list of team names to assign to the Situation. Specify an empty list to deassign all teams from the Situation.

Response

Method **`assignTeamsToSituation`** returns the following response:

Type	Description
JSON Object	A Javascript object containing a list of the team names or team IDs assigned to the Situation, depending on the input.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`assignTeamsToSituation`**:

Example assigning team IDs to Situation

Example request to assign team IDs 1 and 2 to Situation ID 1:

```
var assignTeamIDs = moogdb.assignTeamsToSituation(1, { "team_ids" : [1, 2] } )
```

Example response returning that team IDs 1 and 2 have successfully been assigned to the Situation:

```
{ "team_ids" : [1, 2] }
```

Example assigning team names to Situation

Example request to assign teams Team1 and Team2 to Situation ID 1:

```
var assignTeamNames = moogdb.assignTeamsToSituation(2, { "team_names" : [
"Team1", "Team2" ] } )
```

Example response returning that teams Team1 and Team2 have successfully been assigned to the Situation:

```
{ "team_names" : [ "Team1", "Team2" ] }
```

Example unassigning all teams from a Situation

Example request to unassign all teams from Situation ID 1:

```
var unassignTeamIDs = moogdb.assignTeamsToSituation(1, { "team_ids" : [] } )
```

Example response returning that all teams have successfully been unassigned from the Situation:

```
{ "team_ids" : [] }
```

checkSituationFlag

A MoogDb v2 method that checks whether a flag is associated with a Situation.

See Situation Flags for more information on Cisco Crosswork Situation Manager Situation flags.

Request arguments

Method **`checkSituationFlag`** takes the following request arguments:

Name	Type	Required	Description
`sitn_id`	Number	Yes	ID of the Situation to check.

flag	String	Yes	Name of the flag to check for the specified Situation ID.

Response

Method **checkSituationFlag** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **checkSituationFlag**:

Request example

Example request to check whether Situation 1 contains the flag S1:

```
var result = JSON.stringify(moogdb.checkSituationFlag(1, "S1"))
```

Response example

Example response returning **true** because the Situation contains the specified flags:

```
true
```

closeAlert

A MoogDb v2 method that closes one or more alerts.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **closeAlert** takes the following request arguments:

Name	Type	Required	Description
alertId	Number	No, if you specify **alertIds**.	A single alert ID. You must provide a single **alertId** or a list of **alertIds**.
alertIds	JSON List	No, if you specify	A list of alert IDs. You must provide a single **alertId** or a list of **alertIds**.

		alertId.	
thread_entry_comment	String	No	Thread entry comment you want to add to the closed alert. HTML and XML tags are stripped from the thread entry text. Cisco Crosswork Situation Manager converts reserved characters to HTML entities, for example, **&** is converted to **&**.

Response

Method **closeAlert** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **closeAlert**:

Request example

Example request to close alerts 78, 234, and 737 with a thread entry comment:

```
var success = moogdb.closeAlert([78,234,737], "Closing as agreed during team
discussion 1/1/2018");
```

Response example

A successful request returns **true**.

closeSituation

A MoogDb v2 method that closes a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **closeSituation** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	Situation ID.
closeAlerts	Constant	Yes	Determines how the alerts in the Situation are treated. One of: **CLOSE_NO_ALERT**: No alerts are closed.

			CLOSE_ALL_ALERTS: All alerts are closed. **CLOSE_UNUSED_ALERTS**: Only alerts that are unique to this Situation are closed. To access these constants from a MooBot, precede them with the module name, for example: **moogdb.CLOSE_NO_ALERT**

Response

Method **closeSituation** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **closeSituation**:

Response example

A successful request returns **true**.

createAlert

A MoogDb v2 method that creates or updates an alert. Optionally updates custom info for deduplicated alerts.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **createAlert** takes the following request arguments:

Name	Type	Required	Description
alert	Native Object	Yes	A Javascript object containing alert attributes, such as **type**, **severity**, etc.
event	CEvent	Yes	A CEvent object representing the alert, containing alert

			attributes, such as **type**, **severity**, etc.
mergeCustomInfo	Boolean	No	Set this to **true** to merge the **custom_info** data in this alert with the custom info held in the database.

Response

Method **createAlert** returns the following response:

Type	Description
CEvent	A CEvent object containing the latest version of the alert.

createMaintenanceWindow

A MoogDb v2 method that creates a maintenance window, by passing an object containing the information. A maintenance window filters alerts caused by a known period of maintenance.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **createMaintenanceWindow** takes the following request arguments:

Name	Type	Required	Description
maintenanceWindowObj	Object	Yes	A map containing the following information.

The object **maintenanceWindowObj** contains the following information:

Name	Type	Required	Description
name	String	Yes	Name of the maintenance window.
description	String	Yes	Description of the maintenance window.
filter	String	Yes	An SQL-like filter that alerts must match to be included in the maintenance window.
start_date_time	Number (Epoch)	Yes	Start time of the maintenance window. This must be in Unix epoch time and may be up to 5 years in the future.
duration	Number (Epoch)	Yes	Duration of the maintenance window in seconds. The minimum duration is 1 second and the maximum is 157784630 seconds (5 years).
forward_alerts	Boolean	Yes	Whether or not alerts should be forwarded to the next Moolet in the processing chain.
recurring_period	Number	No	Whether or not this is a recurring maintenance window. Set this to:**1** for a recurring maintenance window.**0** for a one-time maintenance window.If not specified, default is **0**. If you set this property to **1**, you must specify **recurring_period_units**.
recurring_period_unit	Number	No	Specifies the recurring period of the maintenance window, in days, weeks or months. Valid values are:**2** = daily**3** = weekly**4** = monthlyDefault is **0** if **recurring_period** is set to **0**.

Response

Method **createMaintenanceWindow** returns the following response:

Type	Description
Long	ID of the new maintenance window, or null if an error occurred.

Examples

The following examples demonstrate typical use of method **createMaintenanceWindow**:

Request example

Example request to create a new maintenance window which recurs daily:

```
{
    "name": "Phil",
    "description": "A description",
    "filter": "custom_info.eventDetails.alertGroup = Websphere AND source =
my_source12345",
    "start_date_time": 1497971059,
    "duration": 360000,
    "forward_alerts": true,
    "recurring_period": 1,
    "recurring_period_units": 2
}
```

createSituation

A MoogDb v2 method that creates a new Situation, containing no alerts.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **createSituation** takes the following request arguments:

Name	Type	Required	Description
moderator	String	Yes	A valid user name.
label	String		New Situation description.

Response

Method **createSituation** returns the following response:

Type	Description
CEvent	A CEvent object containing the newly created Situation.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes

Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

createTeam

A MoogDb v2 method that creates a new team, by passing an object containing team information.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`createTeam`** takes the following request arguments:

Name	Type	Required	Description
`teamObj`	Object	Yes	An object containing the following information.

The object **`teamObj`** contains the following information:

Name	Type	Required	Description
`name`	String	Yes	New team name. Must be unique.
`alert_filter`	JSON Object	No	An SQL-like filter that alerts must match to be assigned to the team.
`sig_filter`	JSON Object	No	An SQL-like filter that Situations must match to be assigned to the team.
`active`	Boolean	No	Set to **`true`** if the team is active; set to **`false`** if the team is inactive. Default is **`true`**.
`services`	Array of Numbers or Strings	No	List of the team service names or IDs.
`users`	Array of Numbers or Strings	No	List of users in the team, either usernames or IDs.
`description`	String	No	Team description.
`landing_page`	String	No	Default landing page for the team.

Response

Method **`createTeam`** returns the following response:

Type	Description
`Integer`	ID of the new team, or null if an error occurred.

createThread

A MoogDb v2 method that creates a new thread for a Situation. Threads are comments or 'story activity' on Situations.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`createThread`** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	ID of the Situation you want to create a new thread for.
thread	String	Yes	Name of the new thread.

Response

Method **createThread** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

createThreadEntry

Note

This method has been superseded. Use addThreadEntry instead. All new functionality will be delivered in **addThreadEntry**.

A MoogDb v2 method that creates a new entry on an existing thread in a Situation. Threads are comments or 'story activity' on Situations.

This method returns a Boolean indicating whether or not the thread entry was created successfully.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **createThreadEntry** takes the following request arguments:

Name	Type	Required	Description
entry	String	Yes	Description of the new entry you want to create in the thread. For example, "And another thing...". HTML and XML tags are stripped from the thread entry text. Reserved characters are converted to HTML entities, for example, **&** is converted to **&**.
thread	String	Yes	Name of the existing thread.
userId	Number		A valid user ID.
situationId	String	Yes	Situation ID.

Response

Method **`createThreadEntry`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

createUser

A MoogDb v2 method that creates a user, by passing an object containing user properties.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`createUser`** takes the following request arguments:

Name	Type	Required	Description
`userObj`	Object	Yes	An object containing the following user information.
`username`	String	Yes	New user login name. Must be unique.
`password`	String	Yes	New user password. Only valid for DB realm.
`active`	Boolean	No	Set to **`true`** if user is active; set to **`false`** if user is inactive. Default is **`true`**.
`email`	String	Yes	User's email address.
`fullname`	String	Yes	User's full name.
`roles`	JSON Array	Yes	List of either the roleIDs or role names. For example, **`"roles":["Super User"]`**.
`primary_group`	String or Number	Yes	User's primary group name or primary group ID.
`department`	String or Number	Yes	User's department name or ID.
`joined`	Number	Yes	Time the user joined in Unix epoch time.
`timezone`	String	Yes	User's timezone.
`contact_num`	String	Yes	User's phone number.
`session_expiry`	Number	No	Number of minutes after which the user's session

			expires. Default is the system default.
teams	JSON Array of Numbers or Strings	Yes	List of the user's team names or team IDs.

Response

Method **createUser** returns the following response:

Type	Description
Integer	ID of the new user, or null if an error occurred.

Examples

The following examples demonstrate typical use of method **createUser**:

Request example

Example request to create a new user "user1":

```
{
    "username": "phil",
    "fullname": "Phil Customer",
    "roles": ["Super User"],
    "department": 3,
    "active": true,
    "email": "phil@example.com",
    "timezone": "(GMT 00:00) Europe/London - Greenwich Mean Time",
    "teams": [1, 2, 4],
    "joined": 12345678,
    "contact_num": "0965412345"
}
```

Response example

Example response returning the ID of the new user:

```
72
```

createWorkflow

A MoogDb v2 method that creates a new workflow at the end of a Workflow Engine Moolet sequence. To move it, use reorderWorkflows.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **createWorkflow** takes the following request arguments:

Name	Type	Required	Description
moolet_name	String	Yes	Name of the Workflow Engine Moolet that the new workflow belongs to.
workflow_name	String	Yes	Name of the new workflow. Must be unique.

<table>
<tr><td>description</td><td>String</td><td>No</td><td>Description of the new workflow.</td></tr>
<tr><td>entry_filter</td><td>JSON Object</td><td>No</td><td>An SQL-like filter to determine which events, alerts or Situations can enter the workflow. Leave empty for the workflow to accept all events, alerts or Situations.</td></tr>
<tr><td>sweep_up_filter</td><td>JSON Object</td><td>No</td><td>An SQL-like filter to intake any additional events, alerts or Situations from the database.</td></tr>
<tr><td>first_match_only</td><td>Boolean</td><td>Yes</td><td>If enabled, events, alerts, and Situations only pass through actions on the first time they enter this workflow.</td></tr>
<tr><td>operations</td><td>JSON Array</td><td>Yes</td><td>List of properties relating to each operation:
<table>
<tr><th>Name</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>type</td><td>String</td><td>Yes</td><td>Type of operation. Options are: 'action', 'decision' and 'delay'.</td></tr>
<tr><td>operation_name</td><td>String</td><td>Yes, for 'action' and 'decision' types.</td><td>Name of the operation.</td></tr>
<tr><td>function_name</td><td>String</td><td>Yes, for 'action' and 'decision' types.</td><td>Name of the function.</td></tr>
<tr><td>forwarding_behavior</td><td>String</td><td>No</td><td>Forwarding behavior for the function. One of:always forward: The function always forwards the object to the next workflow.stop this workflow: The function stops this workflow and the object moves to the next workflow.stop all workflows: The function stops all workflows for this object.Default is always forward. Only valid for 'action' and 'decision'</td></tr>
</table>
</td></tr>
</table>

						types.
			function_args	JSON Object	No	Arguments for the function.
			duration	Integer	Yes, for 'delay' type.	Length of time before the message goes to the next operation.
			reset	Boolean	Yes, for 'delay' type.	Determines whether the timer resets after each occurrence. Not available if you have set a workflow to **`first_match_only`**. The timer is reset only if an occurrence with the same ID is received (alert_id or situation_id) within the current 'delay' timeframe.

Examples

The following examples demonstrate typical use of method **`createWorkflow`**:

Request example

Example request to create a new workflow "ChangeInfoWorkflow":

```
var id = moogdb.createWorkflow(
{
    "moolet_name": "Alerts Workflows",
    "workflow_name": "ChangeInfoWorkflow",
    "description": "Changingthealertinformation",
    "entry_filter": "((category = \"Closed\") AND (custom_info.test = \"test\"))
AND (description = \"test\")",
    "sweep_up_filter": "((sig_id = 1) AND (first_event_time = 1574121600)) AND
(description = \"test\")",
    "first_match_only": false,
    "operations": [{
        "type": "action",
        "function_name": "functionA",
        "function_args": {"admin": 2},
        "operation_name": "do something"
         },
         {
             "type": "delay",
             "delay": 30,
```

```
                "reset": false
        }]
});
```

deassignAlert

A MoogDb v2 method that removes the assignment of the current owner from an alert, and leaves it unassigned.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`deassignAlert`** takes the following request arguments:

Name	Type	Required	Description
`alertId`	Number	Yes	ID of the alert that you want to deassign the owner from.

Response

Method **`deassignAlert`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

deleteMaintenanceWindow

A MoogDb v2 method that deletes a single maintenance window.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`deleteMaintenanceWindow`** takes the following request arguments:

Name	Type	Required	Description
`maintenanceWindowId`	Number	Yes	ID of the maintenance window you want to delete.

Response

Method **`deleteMaintenanceWindow`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

Examples

The following examples demonstrate typical use of method **deleteMaintenanceWindow**:

Request example

Example request to maintenance window 456:

```
var success = moogdb.deleteMaintenanceWindow(456)
```

Response example

A successful request returns **true**.

deleteMaintenanceWindows

A MoogDb v2 method that deletes maintenance windows that match a filter.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **deleteMaintenanceWindows** takes the following request arguments:

Name	Type	Required	Description
filter	String	Yes	An SQL-like or JSON filter to match maintenance windows that you want to delete.
limit	Number	No	Maximum number of windows to fetch. Default is 100.

Response

Method **deleteMaintenanceWindows** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

Examples

The following examples demonstrate typical use of method **deleteMaintenanceWindows**:

Request examples

Example request to delete maintenance windows that match a filter:

```
var success = deleteMaintenanceWindows(filter, limit);
```

JSON filter where the description is "host375":

```
{ "column": "description", "op": 10, "value": "host375", "type": "LEAF" }
```

Advanced SQL filter where the description is "host375":

```
Description MATCHES "host375"
```

Response example

A successful request returns **true**.

deleteTeam

A MoogDb v2 method that deletes a single team.

Request arguments

Method **`deleteTeam`** takes the following request arguments:

Name	Type	Required	Description
`team_id`	Number	Yes	ID of the team you want to delete.

Response

Method **`deleteTeam`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

Examples

The following examples demonstrate typical use of method **`deleteTeam`**:

Request example

Example request to delete a team with ID 33.

```
var success = moogdb.deleteTeam(33)
```

Response example

A successful request returns **`true`**.

deleteWorkflow

A MoogDb v2 method that deletes a workflow from a Workflow Engine Moolet.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`deleteWorkflow`** takes the following request arguments:

Name	Type	Required	Description
`id`	Integer	Yes	ID of the workflow to delete.

Response

Method **`deleteWorkflow`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

findMaintenanceWindows

A MoogDb v2 method that finds maintenance windows based on a filter and a limit.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`findMaintenanceWindows`** takes the following request arguments:

Name	Type	Required	Description
filter	String	Yes	SQL-like filter to match maintenance windows that you want to find. For example: **description matches "server_45"**.
limit	Number	No	Maximum number of windows to return. Default is 100.

Response

Method **findMaintenanceWindows** returns the following response:

Type	Description
Object	A JSON object containing maintenance windows that match the filter.

Examples

The following examples demonstrate typical use of method **findMaintenanceWindows**:

Request example

Example request to return maintenance windows where the description matches "maintenance":

```
var response = moogdb.findMaintenanceWindows("description matches maintenance");
```

Response example

Example response returning two maintenance windows where the description matches "maintenance":


```
[
  {
    "del_flag": false,
    "forward_alerts": false,
    "last_updated": 1573833276,
    "timezone": "Europe/London",
    "description": "This is my first maintenance window",
    "recurring_period_units": 0,
    "filter": "description MATCHES \"Test\"",
    "duration": 3600,
    "recurring_period": 0,
    "name": "My Maintenance Window 1",
    "updated_by": 3,
    "id": 1,
    "start_date_time": 1573833229
  },
  {
    "del_flag": false,
    "forward_alerts": false,
    "last_updated": 1574164385,
    "timezone": "Europe/London",
    "description": "This is my second maintenance window",
    "recurring_period_units": 0,
    "filter": "(severity IN (0, 1, 2, 3, 4, 5)) AND (owner IN (3))",
    "duration": 3600,
    "recurring_period": 0,
    "name": "My Maintenance Window 2",
    "updated_by": 3,
    "id": 2,
    "start_date_time": 1574164339
```

```
  }
]
```

getActiveSituationIds

A MoogDb v2 method that returns the total number of active Situations, and a list of their Situation IDs. Active Situations are those that are not Closed, Resolved or Dormant.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getActiveSituationIds`** takes no request arguments.

Response

Method **`getActiveSituationIds`** returns the following response:

Type	Description
Native Object	A JSON object containing the total number of Situations returned and the Situation IDs.

Examples

The following examples demonstrate typical use of method **`getActiveSituationIds`**:

Response example

Example response returning ten active Situations:

```
{
     "total_situations":10,
     "sitn_ids":[4, 5, 6, 12, 14, 15, 16, 17, 18, 19]
}
```

getAlert

A MoogDb v2 method that returns details of an alert.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getAlert`** takes either the **`alertID`** or **`signature`**request arguments:

Name	Type	Required	Description
`alertId`	Number	Yes if no signature provided	Alert ID.
singature	String	Yes if no alertId provided	Alert signature.

Response

Method **`getAlert`** returns the following response:

Type	Description
CEvent	A CEvent object containing the alert attributes, such as **`type`** and **`severity`**.

getAlertActions

A MoogDb v2 method that returns the actions for one or more alerts or for a time period.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getAlertActions** takes the following request arguments:

Name	Type	Required	Description
alert_ids	JSON Array of Numbers	No	List of alert IDs.
start	Number	Yes	Starting row from which data should be included.
limit	Number	Yes	Maximum number of actions you want to return.
actions	Array of Numbers	No	List of action codes. If no action codes are specified, all action codes are returned. See Alert Action Codes for a list of action codes and their descriptions. Only action codes 8 (Alert Resolved) and 9 (Alert Closed) are valid.
from	Number	No	Start time of the period you want to retrieve alert actions for. This is in Unix epoch time in seconds.
to	Number	No	End time of the period you want to retrieve alert actions for. This is in Unix epoch time in seconds.

Response

Method **getAlertActions** returns the following response:

Type	Description
Native Object	A JSON object containing the alert action information.

Examples

The following examples demonstrate typical use of method **getAlertActions**:

Request examples

Example request to return alert actions:

```
var actions = moogdb.getAlertActions(request);
```

Example **request** object to return the first 100 actions for alert IDs 1 and 2 for action codes 8 and 9:

```
{
  "alert_ids" : [1, 2],
  "start": 0 ,
  "limit" : 100,
  "actions" : [8, 9]
}
```

Example **request** object to return the first 100 actions for alert IDs 1 and 2 for action codes 8 and 9 between the Unix epoch times 1553861746 and 1553872546:

```
{
            "alert_ids" : [1, 2],
            "limit" : 100,
            "actions" : [8, 9],
            "from" : 1553861746,
```

```
            "to" : 1553872546
    }
```

Response example

Example response returning alert actions with action codes 8 (Alert Resolved) and 9 (Alert Closed):

```
[{
    "uid": 49,
    "action_code": 8,
    "description": "Alert Resolved",
    "details": {},
    "alert_id": 1,
    "timed_at": 1557504393
    }, {
    "uid": 49,
    "action_code": 9,
    "description": "Alert Closed",
    "details": {},
    "alert_id": 1,
    "timed_at": 1557504912
    }
}]
```

getAlertCustomInfo

A MoogDb v2 method that returns custom information for an alert.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getAlertCustomInfo`** takes the following request arguments:

Name	Type	Required	Description
`alertId`	Number	Yes	ID of the alert you want to return custom info data for.
`key`	String	No	Specify the key if you are interested in a specific value. Otherwise the method returns all custom info.

Response

Method **`getAlertCustomInfo`** returns the following response:

Type	Description
Number, List, String, or Object	A map of name-value pairs containing the custom info for the specified alert.

getAlertIds

A MoogDb v2 method that returns the total number of alerts, and a list of the alert IDs, for an alert filter and a limit.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getAlertIds`** takes the following request arguments:

Name	Type	Required	Description

query	JSON Object	Yes	An SQL-like filter that alerts must match to be returned.
limit	Number		Maximum number of alert IDs to return.

Response

Method **getAlertIds** returns the following response:

Type	Description
Native Object	A JSON object containing the total number of alerts and their alert IDs.

Examples

The following examples demonstrate typical use of method **getAlertIds**:

Response example

Example response returning ten alert IDs:

```
{
     "total_alerts":10,
     "alert_ids":[4, 5, 6, 12, 14, 15, 16, 17, 18, 19]
}
```

getAllSessionInfo

A MoogDb v2 method that returns session information for all users over a period of time.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getAllSessionInfo** takes the following request arguments:

Name	Type	Required	Description
from	Number	No	Start time of the period you want to retrieve session information for. This is in Unix epoch time in seconds. If empty, returns all session information for all users.
to	Number	No	End time of the period you want to retrieve session information for. This is in Unix epoch time in seconds. If empty, returns user records to date.
start	Number	No	Starting record from which data should be included. Default is 0, the first record.
limit	Number	No	Maximum number of records you want to return. Default is 200.

Response

Method **getAllSessionInfo** returns the following response:

Type	Description
Number	ID of the session.
String	User name for the session.

Number	Start time of the session, in Unix epoch time.
Number	Last access time within the session, in Unix epoch time.

Examples

The following examples demonstrate typical use of method **`getAllSessionInfo`**:

Request example

Example request to return session information from Unix epoch time 1570544146 to Unix epoch time 1570704144:

```
var UserMap2 =  {"start" : 1, "limit":6, "from": 1570544146, "to":1570704144 };
var SessionInfo2 = moogdb.getAllSessionInfo(UserMap2);
logger.warning("getAllSessionInfo..."+ JSON.stringify(SessionInfo2));
```

Response example

Example response returning session information between Unix epoch times 1570544146 and 1570704144:

```
getAllSessionInfo...
[{"sessionId":2,"userName":"graze","startTime":1570700522,"lastAccess":157070052
2},
{"sessionId":3,"userName":"user4","startTime":1570700529,"lastAccess":1570700529
},
{"sessionId":4,"userName":"admin","startTime":1570700675,"lastAccess":1570700675
},
{"sessionId":5,"userName":"graze","startTime":1570703911,"lastAccess":1570703911
}]
```

getMaintenanceWindows

A MoogDb v2 method that returns maintenance windows based on a start position and a limit. Only returns active recurring windows and scheduled maintenance windows, not expired or deleted maintenance windows.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getMaintenanceWindows`** takes the following request arguments:

Name	Type	Required	Description
`start`	Number	Yes	Number of the first maintenance window to return, 0 to start at the first, 10 to start at the 11th.
`limit`	Number	Yes	Maximum number of maintenance windows to return.

Response

Method **`getMaintenanceWindows`** returns the following response:

Type	Description
Native Object	A JSON object with a nested array containing information on the maintenance windows.

Examples

The following examples demonstrate typical use of method **`getMaintenanceWindows`**:

Request example

Example request to return the first maintenance window in the database:

```
var response = moogdb.getMaintenanceWindows(0,1);
```

Response example

Example response returning details of the first maintenance window:

```
[
  {
    "del_flag": false,
    "forward_alerts": false,
    "last_updated": 1574164385,
    "timezone": "Europe/London",
    "description": "Test",
    "recurring_period_units": 0,
    "filter": "(severity IN (0, 1, 2, 3, 4, 5)) AND (owner IN (3))",
    "duration": 3600,
    "recurring_period": 0,
    "name": "Test",
    "updated_by": 3,
    "id": 2,
    "start_date_time": 1574164339
  }
]
```

getPrcLabels

A MoogDb v2 method that returns Probable Root Cause (PRC) information for all alerts or specified alerts within a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getPrcLabels`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.
`alert_ids`	JSON Array	No	List of alert IDs.

Response

Method **`getPrcLabels`** returns the following response:

Type	Description
Native Object	A JSON object containing the probable root cause information for the alerts in the specified Situation.

Examples

The following examples demonstrate typical use of method **`getPrcLabels`**:

Request example

Example request to return the PRC labels for alerts 1, 2, 3, and 4 in Situation 157:

```
var alertIds = [1,2,3,4];
var prcLabels = moogdb.getPrcLabels(157, alertIds);
```

Response example

Example response returning the PRC labels for alerts 1, 2, 3, and 4 in the Situation:

```
{
    "non_causal": [2,3],
    "unlabelled": [4],
    "causal": [1]
}
```

getProcesses

A MoogDb v2 method that returns a list of processes from the database.

Request arguments

Method **`getProcesses`** takes the following request arguments:

Name	Type	Required	Description
`limit`	Integer	No	Maximum number of processes to return. Default is 1000.
`query`	String	Yes	A JSON or SQL like filter of the process name.
`exact_match`	Boolean	No	If **`true`**, the query performs an exact match on the process name. If **`false`**, the query checks for **`contains`** only on the process name. Default is **`false`**.

Response

Method **`getProcesses`** returns the following response:

Type	Description
Object	A list of strings describing the requested processes, or a null value if there is an error.

Examples

The following examples demonstrate typical use of method **`getProcesses`**:

Request example

Example request to return the first thousand process names containing `"Network"`:

```
var actions = moogdb.getProcesses(1000, "Network", false);
```

Response example

Example response returning returning details of all process names containing `"Network"`:

```
[
    {
        "process_id": 1,
        "name": "Network LON",
        "description": "Network London"
    },
    {
        "process_id": 2,
        "name": "NY Network A",
        "description": "Network New York A"
    },
```

```
    {
        "process_id": 3,
        "name": "NY Network B",
        "description": "Network New York B"
    }
]
```

getQueueName

A MoogDb v2 method that returns the queue name, from the database, for a queue ID.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getQueueName`** takes the following request arguments:

Name	Type	Required	Description
`queueId`	Number	Yes	Queue ID.

Response

Method **`getQueueName`** returns the following response:

Type	Description
String	Queue name for the specified queue ID.

getResolvingThreadEntries

A MoogDb v2 method that returns thread entries for a Situation that have been marked as resolving steps. Threads are comments or 'story activity' on Situations. Operators can mark one or more thread entries as steps that were used to resolve a Situation.

You can select specific thread entries to return using **`start`** and **`limit`** values. If not, their default values return the first 100 thread entries. The thread entries returned are ordered by most recent first.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getResolvingThreadEntries`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.
`thread`	String	Yes	Name of the thread.
`start`	Number	No	Number of the first resolving thread entry to return. Default is 0.
`limit`	Number	No	Maximum number of resolving thread entries to return. Default is 100.

Response

Method **`getResolvingThreadEntries`** returns the following response:

Type	Description

Native Object	A JSON object containing details of the selected thread entries.

Examples

The following examples demonstrate typical use of method **`getResolvingThreadEntries`**:

Request examples

Example request to return the first 100 thread entries that are resolving steps for Situation 58:

```
var resolvingEntries = moogdb.getResolvingThreadEntries(58);
```

Request to return the first 10 thread entries that are resolving steps for Situation 58:

```
var resolvingEntries = moogdb.getResolvingThreadEntries(58, 0, 10);
```

Response example

Example response returning two resolving steps for the specified Situation:

```
{
    "entries": [
    {
        "uid": 3,
        "entry": "A comment",
        "agrees": [],
        "total_comments": 0,
        "thread_id": "Support",
        "mmid": -1,
        "sig_id": 1,
        "entry_id": 2,
        "timed_at": 1423226829,
        "disagrees": [],
        "commenters": []
    },
    {
        "uid": 3,
        "entry": "Another comment",
        "agrees": [],
        "total_comments": 0,
        "thread_id": "Support",
        "mmid": -1,
        "sig_id": 1,
        "entry_id": 1,
        "timed_at": 1423226807,
        "disagrees": [3],
        "commenters": []
    }],
    "total_entries": 2
}
```

getServices

A MoogDb v2 method that returns a list of services from the database.

Request arguments

Method **`getServices`** takes the following request arguments:

Name	Type	Required	Description
`limit`	Integer	No	Maximum number of services to return. Default is 1,000.

start	Integer	No	Number of the first service to return. Default is 0.
query	String	Yes	A JSON or SQL like filter of the service name.
exact_match	Boolean	No	If **true**, the query performs an exact match on the service name. If **false**, the query checks for **contains** only on the service name. Default is **false**.

Response

Method **getServices** returns the following response:

Type	Description
Native Object	A list of strings describing the requested services, or a null value if there is an error.

Examples

The following examples demonstrate typical use of method **getServices**:

Example Using Exact Matching

Example request using exact matching of the query "Network LON":

```
var actions = moogdb.getServices(0, 1000, "Network LON", true);
```

Example response returning details of the service name "Network LON":

```
[{
   "service_id":3,
   "name":"Network LON",
   "description":"Network description"
}]
```

Example using approximate matching

Example request using approximate matching of the query "Network":

```
var actions = moogdb.getServices(0, 1000, "Network", false);
```

Example response returning details of all service names containing "Network":

```
[{
   "service_id":1,
   "name":"Network LON",
   "description":"Network London"
},{
   "service_id":2,
   "name":"NY Network A",
   "description":"Network New York A"
},{
   "service_id":3,
   "name":"NY Network B",
   "description":"Network New York B"
}]
```

getSigCorrelationInfo

A MoogDb v2 method that returns all correlation information related to a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getSigCorrelationInfo** takes the following request argument:

Name	Type	Required	Description
sitn_id	Number	Yes	Situation ID.

Response

Method **getSigCorrelationInfo** returns the following response:

Type	Description
Object	A JSON object containing a list of maps of correlation info.

getSigCustomInfo

A MoogDb v2 method that returns all custom info related to a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getSigCustomInfo** takes the following request arguments:

Name	Type	Required	Description
sigId	Number	Yes	Situation ID.
key	String		Node path for specific value to return.

Response

Method **getSigCustomInfo** returns the following response:

Type	Description
Number, Object, Array or String	Response depends on the key but can be a number, object, array or string containing a list of maps of custom info.

getSituation

A MoogDb v2 method that returns details of a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getSituation** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	Situation ID.

Response

Method **getSituation** returns the following response:

Type	Description
Object	A JSON object representing the Situation.

getSituationActions

A MoogDb v2 method that returns the actions for one or more Situations or for a time period. Created by passing an object with the information requested. You can use the **`from`** and **`to`** arguments to specify a period that you want to retrieve Situation actions for. If you do not specify these, all actions are returned.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getSituationActions`** takes the following request arguments:

Name	Type	Required	Description
`sitn_ids`	JSON Array	No, if **`to`** and **`from`** are used.	Array of Situation IDs that the actions are requested for.
`start`	Number	Yes	Number of the first action to return.
`limit`	Number	Yes	Maximum number of actions that you want to return.
`actions`	JSON Array	No	List of action codes. Returns all action codes if no action codes are specified. See Situation Action Codes for a list of action codes and their descriptions.
`from`	Number	No	Start time of the period you want to retrieve Situation actions for. This is a Unix epoch timestamp in seconds.
`to`	Number	No	End time of the period you want to retrieve Situation actions for. This is a Unix epoch timestamp in seconds.

Response

Method **`getSituationActions`** returns the following response:

Type	Description
Native Object	A JSON object containing the activity for the specified Situations.

Examples

The following examples demonstrate typical use of method **`getSituationActions`**:

Request examples

Example request to return Situation actions:

```
var actions = moogdb.getSituationActions(request);
```

Example **`request`** object to return the first 100 actions for Situation IDs 1, 2, and 3 for action codes 1 (Situation Created) and 14 (Added Alerts To Situation):

```
{
    "sitn_ids" : [1, 2, 3],
    "start" : 0,
    "limit" : 100,
    "actions" : [1, 14]
}
```

Response example

Example response returning the requested actions for the specified Situations:

```
[
    {
        "uid": 2,
        "action_code": 1,
        "description": "Situation Created",
        "details": {},
        "type": "event",
        "sig_id": 1,
        "timed_at": 1507039842
     },
    {
        "uid": 2,
        "action_code": 14,
        "description": "Added Alerts To Situation",
        "details": {},
        "alerts": [1, 2]
    }
]
```

getSituationAlertIds

A MoogDb v2 method that returns the total number of alerts, and a list of their alert IDs, for a Situation. This can be either all alerts or just those alerts unique to the Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getSituationAlertIds`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.
`uniqueOnly`	Boolean	No	Indicates the alerts to return from the Situation:**`true`**: Return only alerts unique to the Situation.**`false`**: Return all alerts in the Situation. Default.

Response

Method **`getSituationAlertIds`** returns the following response:

Type	Description
Native Object	A JSON object containing the total number of alerts and the alert IDs for the specified Situation.

Examples

The following examples demonstrate typical use of method **`getSituationAlertIds`**:

Response example

Example response returning the total number of alerts and the alert IDs for the specified Situation:

```
{
     "total_alerts":10,
     "alert_ids":[4, 5, 6, 12, 14, 15, 16, 17, 18, 19]
}
```

getSituationFlags

A MoogDb v2 method that returns the flags for one or more Situations.

See Situation Flags for more information on Cisco Crosswork Situation Manager Situation flags.

Request arguments

Method **`getSituationFlags`** takes the following request arguments:

Name	Type	Required	Description
`sitn_ids`	Array	Yes	IDs of the Situations to return the flags for.

Response

Method **`getSituationFlags`** returns the following response:

Type	Description
JSON Array	An array of the flags associated with the specified Situation IDs.

Examples

The following examples demonstrate typical use of method **`getSituationFlags`**:

Request example

Example request to return the flags associated with Situation IDs 1 and 2:

```
var result = JSON.stringify(moogdb.getSituationFlags([1, 2]))
```

Response example

Example response returning the flags associated with specified Situations:

```
{"1":["A1", "B1"], "2":["A1"]}
```

getSituationHosts

A MoogDb v2 method that returns a list of host names for a Situation, either for all alerts in the Situation or just for unique alerts.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getSituationHosts`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.
`uniqueOnly`	Boolean	No	Indicates the host names to return from the Situation:**`true`**: Return only host names unique to the Situation.**`false`**: Return all host names in the Situation. Default.

Response

Method **`getSituationHosts`** returns the following response:

Type	Description
Native Object	A JSON array containing the host names.

Examples

The following examples demonstrate typical use of method **`getSituationHosts`**:

Response example

Example response returning the host names for the Situation:

```
{
    "hosts": [
        "server1",
        "server2",
        "server3",
        "server4",
        "server5",
        "server6",
        "server7"
    ]
}
```

getSituationIds

A MoogDb v2 method that returns the total number of Situations, and a list of their Situation IDs, for a Situation filter and a limit.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getSituationIds`** takes the following request arguments:

Name	Type	Required	Description
`query`	JSON Object	Yes	An SQL-like filter that Situations must match to be returned.
`limit`	Number		Maximum number of Situation IDs to return.

Response

Method **`getSituationIds`** returns the following response:

Type	Description
Native Object	A JSON object containing the total and the Situation IDs.

Examples

The following examples demonstrate typical use of method **`getSituationIds`**:

Response example

Example response returning the total number of Situations and the Situation IDS matching the filter:

```
{
     "total_situations":10,
     "sitn_ids":[4, 5, 6, 12, 14, 15, 16, 17, 18, 19]
}
```

getSituationPrimaryTeam

A MoogDb v2 method that returns the primary team assigned to a Situation.

Request arguments

Method **getSituationPrimaryTeam** takes no request arguments.

Name	Type	Required	Description
sitn_id	Number	Yes	ID of the Situation you want to return the primary team for.

Response

Method **getSituationPrimaryTeam** returns the following response:

Type	Description
Object	A Javascript object containing the Situation ID and the primary team ID.

Examples

The following examples demonstrate typical use of method **getSituationPrimaryTeam**:

Request example

Example request to return the primary team for Situation 1906:

```
var actions = moogdb.getSituationPrimaryTeam(1906);
```

Response examples

Example response returning that team 36 is the primary team for Situation 1906:

```
{
    "primary_team_name": "Infrastructure",
    "sitn_id":1906,
    "primary_team_id":36
}
```

Example response returning that Situation 1906 does not have a primary team assigned to it:

```
{
    "sitn_id":1906,
}
```

getSituationProcesses

A MoogDb v2 method that returns a list of process names for a Situation, and the primary process name, if defined.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getSituationProcesses** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	Situation ID.

Response

Method **getSituationProcesses** returns the following response:

Type	Description
Native Object	A JSON array containing the process names, and the Situation's primary process, if defined.

Examples

The following examples demonstrate typical use of method **`getSituationProcesses`**:

Response example

Example response returning the processes and the primary process for a Situation:

```
{
    "processes": ["Process1", "Process2"],
    "primary": "Process2"
}
```

getSituationServices

A MoogDb v2 method that returns a list of service names for a Situation, and the primary service name, if defined.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getSituationServices`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.

Response

Method **`getSituationServices`** returns the following response:

Type	Description
Native Object	A JSON array containing the service names, and the Situation's primary service, if defined.

Examples

The following examples demonstrate typical use of method **`getSituationServices`**:

Response example

Example response returning the services and the primary service for a Situation:

```
{
    "services": ["Service1","Service2"],
    "primary": "Service1"
}
```

getSituationsWithFlag

A MoogDb v2 method that returns all the Situations which have a specified flag.

See Situation Flags for more information on Cisco Crosswork Situation Manager Situation flags.

Request arguments

Method **`getSituationsWithFlag`** takes the following request arguments:

Name	Type	Required	Description

flag	String	Yes	Name of the flag to search for.
start	Number	No	Starting point of the result set to return. Default is 0.
limit	Number	No	Number of results to return. Default is 1000.

Response

Method **getSituationsWithFlag** returns the following response:

Type	Description
JSON Array	An array of the Situation IDs that have the specified flag associated with them.

Examples

The following examples demonstrate typical use of method **getSituationsWithFlag**:

Request example

Example request to return all Situations with flag "S1":

```
var result = JSON.stringify(moogdb.getSituationsWithFlag("S1",0,1000))
```

Response example

Example response returning the Situation IDs with the specified flag:

```
[1, 2, 3]
```

getSituationTopology

A MoogDb v2 method that returns the topology details for a specified Situation and topology. The request returns a JSON object that lists the links and nodes affected by the Situation in a specified topology. It also returns the alert matching attributes for the nodes in the topology.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getSituationTopology** takes the following request arguments:

Name	Type	Required	Description
sigId	Number	Yes	Situation ID.
topologyName	String	Yes	Name of the topology for which to return the Situation's link, node and alert matching attribute details. A Situation can impact nodes in multiple topologies.
context	Integer	Yes	Number, between 0 and 4, of contextual hops from the nodes directly affected within the Situation to other nodes to be included in the returned object. See Vertex Entropy for more information on contextual hops.Vertex Entropy**0**: Only nodes directly affected by the Situation. Default.**4**: Nodes that are up to four hops away from the nodes directly affected by the Situation.
properties	Array of	Yes	List of the node properties to be returned. Valid properties are:**severity**: Severity of the node.**prc**: Whether this node is the

	Strings		probable root cause of the alert.**`description`**: Description of the node.**`vertex_entropy`**: Vertex Entropy of the node. See Vertex Entropy for more information.

Response

The method returns the following response:

Type	Description
Object	A JSON object in NetJSON format that lists the links, nodes and alert matching attributes affected by the Situation.

Successful requests return an array of JSON objects containing the following:

Examples

The following examples demonstrate typical use of method **`getSituationTopology`**:

Request example

Example request to retrieve the topology link and node details for Situation ID 12 in the "host" topology within 4 hops:

```
var sigId = 12;
var context = 4;
var topologyName = "host";
var properties = ["severity", "vertex_entropy", "prc", "description"];
var situationTopology = moogdb.getSituationTopology(sigId, context,
topologyName, properties);
```

Response example

Example response returning the links, nodes and alert matching attributes for the Situation:

```
{
    "links": [
    {
        "source":"appsvr02",
        "target":"appsvr03"
    },
    {
        "source":"appsvr01",
        "target":"sd-01"},
    {
        "source":"appsvr02",
        "target":"appsvr01"},
    {
        "source":"sd-02",
        "target":"sd-01"
    }],
    "nodes": [
    {
        "id":"appsvr02",
        "properties":
        {
            "prc":null,
            "severity":5,
            "context":0,
            "description":"node1",
            "vertex_entropy": 0.1794592472207979
        }
```

```
        },
        {
            "id":"appsvr03",
            "properties":
            {
                "prc":null,
                "severity":null,
                "context":1,
                "description":"node2",
                "vertex_entropy": 0.1794592472207979
            }
        },
        {
            "id":"appsvr01",
            "properties":
            {
                "prc":null,
                "severity":5,
                "context":0,
                "description":"node3",
                "vertex_entropy": 0.08976540495989357
            }
        },
        {
            "id":"sd-02",
            "properties":
            {
                "prc":null,
                "severity":0,
                "context":0,
                "description":"node4",
                "vertex_entropy": 0.08976540495989357
            }
        },
        {
            "id":"sd-01",
            "properties":
            {
                "prc":null,
                "severity":0,
                "context":0,
                "description":"node5",
                "vertex_entropy": 0.1794592472207979
            }
        }],
        "alertMatchingAttributes":["source"]
    }
```


getSituationVisualization

A MoogDb v2 method that returns the Visualize tab information for a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getSituationVisualization`** takes the following request arguments:

Name	Type	Required	Description
id	Number	Yes	Situation ID.

Response

Method **getSituationVisualization** returns the following response:

Type	Description

Examples

The following examples demonstrate typical use of method **getSituationVisualization**:

Request example

Example request to return the Visualize information for Situation 99:

```
var visualizeData = moogdb.getSituationVisualization(99);
```

getTeam

A MoogDb v2 method that returns a team's details by team ID or team name.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getTeam** takes the following request arguments:

Name	Type	Required	Description
team_id	Integer	No, if you specify **name**.	ID of the team to return information about.
name	String	No, if you specify **team_id**.	Name of the team to return information about.

Response

Method **getTeam** returns the following response:

Type	Description
JSON Object	A JSON object containing information about the team.

Examples

The following examples demonstrate typical use of method **getTeams**:

Request example

Example request to return the details of team ID 1:

```
var teamData = moogdb.getTeam(1);
```

Response example

Example response returning information on team ID 1:

```
{
    "room_id": 1,
    "alert_filter": "(severity = 0) AND (severity = 1)",
    "user_ids": [3],
    "sig_filter": "((internal_priority = 0) AND (description = \"Test\")) AND
(last_state_change = 1574121600)",
```

```
    "landing_page": "",
    "description": "",
    "active": true,
    "team_id": 1,
    "services": [],
    "users": ["admin"],
    "deleted": false,
    "name": "Cloud DevOps",
    "service_ids": []
}
```

getTeams

A MoogDb v2 method that returns details of all teams.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getTeams`** takes no request arguments.

Response

Method **`getTeams`** returns the following response:

Type	Description
JSON Object	A JSON object containing information about all teams in Cisco Crosswork Situation Manager.

Examples

The following examples demonstrate typical use of method **`getTeams`**:

Request example

Example request to return details of all teams:

```
var response = moogdb.getTeams();
```

Response example

Example response returning information on all teams in Cisco Crosswork Situation Manager:

```
[{
    "room_id": 1,
    "alert_filter": "(severity = 0) AND (severity = 1)",
    "user_ids": [3],
    "sig_filter": "((internal_priority = 0) AND (description = \"Test\")) AND
(last_state_change = 1574121600)",
    "landing_page": "",
    "description": "",
    "active": true,
    "team_id": 1,
    "services": [],
    "users": ["admin"],
    "deleted": false,
    "name": "Cloud DevOps",
    "service_ids": []
```

```
    },
    {
      "room_id": 2,
      "alert_filter": "",
      "user_ids": [5,6,7,8,9],
      "sig_filter": "",
      "landing_page": null,
      "description": "Team based in Kingston",
      "active": true,
      "team_id": 2,
      "services": ["Kingston::AD::Server","Kingston::Application::Server"],
      "users": ["AnnaMatthews1","JorgeHowell2","LilyHolt3"],
      "deleted": false,
      "name": "Team Kingston",
      "service_ids": [1,2]
    },
    {
      "room_id": 3,
      "alert_filter": "",
      "user_ids": [10,11,12,13,14],
      "sig_filter": "",
      "landing_page": null,
      "description": "Team based in Waterloo",
      "active": true,
      "team_id": 3,
      "services": ["Waterloo::AD::Server
","Waterloo::Application::Server","Waterloo::Catalog::Server"],
      "users": ["ElizaJordan6","LeslieDiaz7"],
      "deleted": false,
      "name": "Team Waterloo",
      "service_ids": [16,17,18]
    },
    {
      "room_id": 4,
      "alert_filter": "",
      "user_ids": [15,16,17],
      "sig_filter": "",
      "landing_page": null,
      "description": "Team based in Warren Street",
      "active": true,
      "team_id": 4,
      "services": ["Warren Street::AD::Server","Warren
Street::Application::Server"],
      "users": ["MorrisCox12","VictoriaGarcia13"],
      "deleted": false,
      "name": "Team Warren Street",
      "service_ids": [31,32,33]
    }]
```


getTeamsForService

A MoogDb v2 method that returns details of all teams related to a service.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getTeamsForService`** takes the following request arguments:

Name	Type	Required	Description
`service_id`	String	No, if you specify **`name`**.	ID of the service.

name	String	No, if you specify **service_id**.	Name of the service.

Response

Method **getTeamsForService** returns the following response:

Type	Description
JSON	A JSON object containing information about all teams related to the specified services in Cisco Crosswork Situation Manager.

Examples

The following examples demonstrate typical use of method **getTeamsForService**:

Request example

Example request to return all teams related to service ID 1:

```
var response = moogdb.getTeamsForService(1);
```

Response example

Example response returning details of the teams related to the service:

```
[{
    "room_id": 2,
    "alert_filter": "(significance = 3) OR (significance = 0)",
    "user_ids": [5,6,7],
    "sig_filter": "(internal_priority = 1) AND (internal_priority = 2)",
    "landing_page": "",
    "description": "Team based in Kingston",
    "active": true,
    "team_id": 2,
    "services": ["Kingston::AD::Server","Kingston::Application::Server"],
    "users": ["LisaRichards1","TerrencePowell2","JacksonKnight3"],
    "deleted": false,
    "name": "Team Kingston",
    "service_ids": [1,2,3]
}]
```

getTeamSituationIds

A MoogDb v2 method that returns the total number of Situations, and a list of the Situation IDs, associated with a team.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getTeamSituationIds** takes the following request arguments:

Name	Type	Required	Description
teamName	String	Yes	Team name.
limit	Number		Maximum number of Situations to return.

Response

Method **getTeamSituationIds** returns the following response:

Type	Description
JSON Object	A JSON object containing the total number of Situations and the Situation IDs associated with the team.

Examples

The following examples demonstrate typical use of method **getTeamSituationIds**:

Response example

Example response returning the total number of Situations and the Situation IDs associated with the team:

```
{
    "total_situations":10,
    "sitn_ids":[4, 5, 6, 12, 14, 15, 16, 17, 18, 19]
}
```

getThreadEntries

A MoogDb v2 method that returns thread entries for a thread in a Situation. Threads are comments or 'story activity' on Situations.

You can request to return specific thread entries using **start** and **limit** values. If not, their default values return the first 100 entries. The entries returned are ordered by most recent first.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getThreadEntries** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	Situation ID.
thread	String	Yes	Name of the thread.
start	Number	No	Number of the first thread entry to return. Default is 0.
limit	Number	No	Maximum number of thread entries to return. Default is 100.

Response

Method **getThreadEntries** returns the following response:

Type	Description
JSON Object	A JSON object containing details of the requested thread entries.

Examples

The following examples demonstrate typical use of method **getThreadEntries**:

Request example

Example request to return the thread entries for thread "Support" on Situation ID 58:

```
var threadEntries = moogdb.getThreadEntries(58, "Support", 0, 10);
```

Response example

Example response returning two thread entries on the specified thread:


```
{
    "entries": [
    {
        "uid": 3,
        "entry": "A comment",
        "agrees": [],
        "total_comments": 0,
        "thread_id": "Support",
        "mmid": -1,
        "sig_id": 58,
        "entry_id": 2,
        "timed_at": 1423226829,
        "disagrees": [],
        "commenters": []
    },
    {
        "uid": 3,
        "entry": "Another comment",
        "agrees": [],
        "total_comments": 0,
        "thread_id": "Support",
        "mmid": -1,
        "sig_id": 58,
        "entry_id": 1,
        "timed_at": 1423226807,
        "disagrees": [3],
        "commenters": []
    }],
    "total_entries": 2
}
```


getThreadEntry

A MoogDb v2 method that returns a thread entry specified using thread entry ID. Threads are comments or 'story activity' on Situations.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getThreadEntry`** takes the following request arguments:

Name	Type	Required	Description
`entry_id`	Number	Yes	Entry ID.

Response

Method **`getThreadEntry`** returns the following response:

Type	Description
JSON Object	A JSON object containing details of the requested thread entry.

Examples

The following examples demonstrate typical use of method **`getThreadEntry`**:

Request example

Example request to return the thread entry with thread entry ID "1":

```
var result = JSON.stringify(moogdb.getThreadEntry(1));
```

Response example

Example response returning thread entry with thread entry ID "1":

```
{
    "entry_id":1,
    "sig_id":1,
    "thread_id":
    "Support",
    "standard_thread":"Support",
    "status":1,
    "timed_at":1586948533,
    "uid":3,
    "did":1,
    "entry":
    "Entry Text1111",
    "mmid":-1
}
```

getToolShares

A MoogDb v2 method that returns the shared access for a tool.

Request arguments

Method **`getToolShares`** takes the following request arguments:

Name	Type	Required	Description
`tool_id`	Number	Yes	ID of the tool that you want to retrieve its shared access for.

Response

Method **`getToolShares`** returns the following response:

Type	Description
Object	A Javascript object containing the tool ID, the domain, and an array of all the domain IDs that can access the tool.

Examples

The following examples demonstrate typical use of method **`getToolShares`**:

Request example

Example request to retrieve all the domain IDs that have access to tool 15 :

```
var actions = moogdb.getToolShares(15);
```

Response example

Example response returning that tool ID 15 can be accessed by team ID 3:

```
{
    "tool_id": 15,
    "domain_ids": [3],
    "domain": "team"
}
```

getTopPrcDetails

A MoogDb v2 method that returns the top most likely causal alerts, based on their Probable Root Cause value, for a Situation.

You can select the maximum number of causal alerts to return using a limit value. If not specified, the endpoint only returns the alert with the highest root cause probability.

The entries returned are ordered with the highest root cause probability first, for the specified Situation, irrespective of whether they have been labeled causal or are unlabeled. Alerts marked as symptoms are excluded from the return.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getTopPrcDetails`** takes the following request arguments:

Name	Type	Required	Description
`sitn_id`	Integer	Yes	ID of the Situation you want to retrieve the Probable Root Cause details for.
`limit`	Integer	No	Maximum number of causal or unlabeled alerts to return. Default is 1, if not specified, returning one alert with the highest root cause probability.

Response

Method **`getTopPrcDetails`** returns the following response:

Type	Description
JSON Array	An array of objects containing the details of the causal or unlabeled alerts with the highest root cause probability in the specified Situation.

The following details are returned for each alert:

Name	Type	Description
`rc_probability`	Number	Root cause probability of the alert.
`description`	String	Description of the alert.
`rc_label`	Integer	Label defining whether the alert is causal or unlabeled. Alerts marked as symptoms are excluded from the return. 1 = causal 0 = unlabeled -1 = symptom
`alert_id`	Integer	Alert ID.

Examples

The following examples demonstrate typical use of method **`getTopPrcDetails`**:

Request example

Example request to return the top three causal alerts with the highest root cause probability in Situation 145:

```
var result = JSON.stringify(moogdb.getTopPrcDetails(145,3))
```

Response example

Example response returning the top three causal or unlabeled alerts for Situation ID 145:

```
{
    "alerts": [
    {
        "rc_probability":0.9933107459030244,
        "description":"Web Server HTTPD is DOWN",
        "rc_label":1,
        "alert_id":53
    },
    {
        "rc_probability":0.9933092393241993,
        "description":"Web Server HTTPD is DOWN",
        "rc_label":1,
        "alert_id":8
    },
    {
        "rc_probability":0.22480057080448923,
        "description":"Web Server HTTPD is DOWN",
        "rc_label":0,
        "alert_id":39
    }]
}
```

getUser

A MoogDb v2 method that returns details of a user.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getUser`** takes the following request arguments:

Name	Type	Required	Description
`userId`	Number	No, if you specify **`username`**.	A valid user ID.
`username`	String	No, if you specify **`userId`**.	A valid username.

Response

Method **`getUser`** returns the following response:

Type	Description
CEvent	A CEvent object containing the user information.

Examples

The following examples demonstrate typical use of method **`getUser`**:

Request example

Example request to return the details of user ID 6:

```
var cevent = moogdb.getUser(6);
```

Response example

Example response returning the details of user ID 6:

```
{
    active=true,
    competencies=[],
    contact_num=,
    department=null,
    description=Online,
    email=customer@example.com,
    fullname=cyber,
    groupname=End-User,
    invitations=[],
    joined=1516963803,
    only_ldap=0,
    photo=-1,
    primary_group=1,
    profile_image=null,
    realms=[DB],
    roles=[1, 3, 4, 5],
    session_expiry=null,
    status=1,
    teams=[],
    timezone=SYSTEM,
    uid=6,
    username=phil
}
```

getUserName

A MoogDb v2 method that returns the username for a user ID.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getUserName`** takes the following request arguments:

Name	Type	Required	Description
`userId`	Number	Yes	A valid user ID.

Response

Method **`getUserName`** returns the following response:

Type	Description
String	The corresponding username for the requested user ID.

getUserRoles

A MoogDb v2 method that returns the roles of a user.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getUserRoles** takes the following request arguments:

Name	Type	Required	Description
userId	Number	No, if you use **username**.	A valid user ID.
username	String	No, if you use **userId**.	A valid username.

Response

Method **getUserRoles** returns the following response:

Type	Description
JSON Object	A JavaScript object containing the role IDs, role names and role descriptions for the specified user.

Examples

The following examples demonstrate typical use of method **getUserRoles**:

Request example

Example request to return the user roles for user ID 6:

```
var cevent = moogdb.getUserRoles(6);
```

Response example

Example response returning the roles of the specified user:

```
[{
    "id": 1,
    "name": "Super User",
    "description": "Super User"
},
{
    "id": 3,
    "name": "Manager",
    "description": "Manager"
},
{
    "id": 4,
    "name": "Operator",
    "description": "Operator"
}]
```

getUsers

A MoogDb v2 method that returns details of all users.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getUsers** takes the following request arguments:

Name	Type	Required	Description
limit	Number	No	Maximum number of users to return. Default is 1000.

Response

Method **getUsers** returns the following response:

Type	Description
JSON Object	A JSON object containing information on all users, up to the limit.

Examples

The following examples demonstrate typical use of method **`getUsers`**:

Request example

Example request to return the details of all users:

```
var cevent = moogdb.getUsers;
```

Response example

Example response returning the details of all users:

```
[{
    "uid": 3,
    "teams": ["Cloud DevOps"],
    "fullname": "Administrator",
    "username": "admin"
},
{
    "uid": 6,
    "teams": [],
    "fullname": "Nagios",
    "username": "Nagios"
},
{
    "uid": 5,
    "teams": [],
    "fullname": "Webhook",
    "username": "Webhook"
}]
```

getUserSessionInfo

A MoogDb v2 method that returns session information for a single user over a period of time.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getUserSessionInfo`** takes the following request arguments:

Name	Type	Required	Description
`username`	String	Yes	Name of the user.
`from`	Number	No	Start time of the period you want to retrieve session information for. This is in Unix epoch time in seconds. If empty, returns all session information for the user.
`to`	Number	No	End time of the period you want to retrieve session information for. This is in Unix epoch time in seconds. If empty, returns user records to date.
`start`	Number	No	Starting record from which data should be included. Default is 0, the first

			record.
limit	Number	No	Maximum number of records you want to return. Default is 200.

Response

Method **getUserSessionInfo** returns the following response:

Type	Description
Number	ID of the session.
Number	Start time of the session, in Unix epoch time.
Number	Last access time within the session, in Unix epoch time.

Examples

The following examples demonstrate typical use of method **getUserSessionInfo**:

Request example

Example request to return ten records of session information for username "graze", starting from the second session:

```
var UserMap =  {"username":"graze", "from":1570544146, "to":1570700529,
"start":2, "limit":10};
var SessionInfo = moogdb.getUserSessionInfo(UserMap);
logger.warning("getUserSessionInfo with username, start, limit, from and to
..."+ JSON.stringify(SessionInfo));
```

Response example

Example response returning session information for username "graze":

```
getUserSessionInfo with username, start, limit, from and to ...
[
    {"sessionId":2,"startTime":1570700522,"lastAccess":1570700522},
    {"sessionId":3,"startTime":1570700529,"lastAccess":1570700529},
    {"sessionId":7,"startTime":1570700675,"lastAccess":1570700675},
    {"sessionId":9,"startTime":1570703911,"lastAccess":1570703911},
    {"sessionId":13,"startTime":1570704735,"lastAccess":1570704735}
]
```

getUserTeams

A MoogDb v2 method that returns the team IDs and team names for a user.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **getUserTeams** takes the following request arguments:

Name	Type	Required	Description
userId	Number	No, if you specify **username**.	A valid user ID.
username	String	No, if you specify **userId**.	A valid username.

Response

Method **getUserTeams** returns the following response:

Type	Description
CEvent	A CEvent object containing the team IDs and team names for the specified user.

Examples

The following examples demonstrate typical use of method **`getUserTeams`**:

Request example

Example request to return the details of user ID 6:

```
var cevent = moogdb.getUserTeams(6);
```

Response example

Example response returning the team IDs and team names for user ID 6:

```
[{
    "id": 2,
    "name": "Alpha"
},
{
    "id": 3,
    "name": "Epsilon"
},
{
    "id": 4,
    "name": "Delta"
}]
```

getWorkflowEngineMoolets

A MoogDb v2 method that returns a list of Workflow Engine Moolets and the functions available in each. This endpoint returns an empty list if Moogfarmd is not running.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getWorkflowEngineMoolets`** takes no request arguments.

Response

Method **`getWorkflowEngineMoolets`** returns a JSON array of Workflow Engine Moolet objects. Each object has the following:

Name	Type	Description
`moolet_name`	String	Workflow Engine Moolet name.
`moolet_type`	ENUM/String	Workflow Engine Moolet type: event, alert, or Situation.
`active`	Boolean	Whether or not the workflow engine that the Moolet represents is active.
`functions[c][d][e]`	JSON	The available functions in the Workflow Engine Moobot - each key is the function name and the values are:

		Description (String): Description of the function. Decision (Boolean): If **`true`**, treat the result of this function as a decision. Arguments (JSON): Arguments of the function, a map from the argument name to: Type (ENUM/String): Type of argument - either Text, JSON or Number. Description: Human readable description of the argument.
`last_updated`	Number	Time when the Workflow Engine Moolet was last updated, in Unix Epoch time.

Examples

The following examples demonstrate typical use of method **`getWorkflowEngineMoolets`**:

Request example

Example request to return information on all of the Workflow Engine Moolets in Cisco Crosswork Situation Manager:

```
var workflowEngineMoolets = moogdb.getWorkflowEngineMoolets;
```

Response example

Example response returning information on all of the Workflow Engine Moolets in Cisco Crosswork Situation Manager:

```
[{
    "active": true,
    "last_updated": 1567420771,
    "moolet_name": "Alert Workflows",
    "functions": {
        "alertInSituation": {
            "decision": true,
            "validators": null,
            "name": "alertInSituation",
            "description": "Check if the alert is in an active Situation.",
            "arguments": [],
            "actionOnAssociated": true,
            "type": ["alert"]
        },
        "alertNotInSituation": {
            "decision": true,
            "validators": null,
            "name": "alertNotInSituation",
            "description": "Check if the alert is not in an active Situation.",
            "arguments": [],
            "actionOnAssociated": true,
            "type": ["alert"]
        },
        "between": {
            "decision": true,
            "validators": null,
            "name": "between",
            "description": "Check to see if the trigger falls between two times,
and optionally on specific days.",
            "arguments": [
```

```
                {
                    "name": "from",
                    "validator": {
                    "regex": "^[0-9]{2}:[0-9]{2}:[0-9]{2}$"
                },
                    "description": "The 'from' time in hh:mm:ss 24hr format",
                    "type": "string",
                    "required": true
                },
                {
                    "name": "to",
                    "validator": {
                        "regex": "^[0-9]{2}:[0-9]{2}:[0-9]{2}$"
                    },
                    "description": "The 'to' time in hh:mm:ss 24hr format",
                    "type": "string",
                    "required": true
                },
                {
                    "name": "days",
                    "description": "The optional list of days in short form
(Mon,Tue,Wed...), for all days use a blank list []",
                    "type": "object",
                    "required": true
                }
            ],
            "actionOnAssociated": false,
            "type": ["alert","situation"]
        },
        "contains": {
            "decision": true,
            "validators": null,
            "name": "contains",
            "description": "Check whether the specified object field contains
any of the listed values. Define values as an array, for example [ a ] or [ a,
b, c ].",
            "arguments": [
                {
                    "name": "field",
                    "description": "The name of the object field to check values
in (including custom_info).",
                    "type": "string",
                    "required": true
                },
                {
                    "name": "values",
                    "description": "The list of values to check for, any
intersection is valid.",
                     "type": "object",
                    "required": true
                }
            ],
            "actionOnAssociated": true,
            "type": ["event","alert","situation"]
        },
        "containsAll": {
            "decision": true,
```

```
            "validators": null,
            "name": "containsAll",
            "description": "Check whether the specified object field contains
all of the listed values. Define values as an array, for example [ a ] or [ a,
b, c ].",
            "arguments": [
                {
                    "name": "field",
                    "description": "The name of the object field to check values
in (including custom_info).",
                    "type": "string",
                    "required": true
                },
                {
                    "name": "values",
                    "description": "The list of values to check for, all must be
included to be valid.",
                    "type": "object",
                    "required": true
                }
            ],
            "actionOnAssociated": true,
            "type": ["event","alert","situation"]
        },
        "doesNotContain": {
            "decision": true,
            "validators": null,
            "name": "doesNotContain",
            "description": "Check whether the specified object field does not
contain any of the listed values. Define values as an array, for example [ a ]
or [ a, b, c ].",
             "arguments": [
                {
                    "name": "field",
                    "description": "The name of the object field to check values
in (including custom_info).",
                    "type": "string",
                    "required": true
                },
                {
                    "name": "values",
                    "description": "The list of values to check for, any
intersection will count.",
                    "type": "object",
                    "required": true
                }
            ],
            "actionOnAssociated": true,
            "type": ["event","alert","situation"]
        }
    },
       "moolet_type": "alert"
},
{
    "active": true,
    "last_updated": 1567420777,
    "moolet_name": "Enrichment Workflows",
    "functions": {
        "alertInSituation": {
            "decision": true,
            "validators": null,
```

```
            "name": "alertInSituation",
            "description": "Check if the alert is in an active Situation.",
            "arguments": [],
            "actionOnAssociated": true,
            "type": ["alert"]
        },
        "alertNotInSituation": {
            "decision": true,
            "validators": null,
            "name": "alertNotInSituation",
            "description": "Check if the alert is not in an active Situation.",
            "arguments": [],
            "actionOnAssociated": true,
            "type": ["alert"]
        },
        "between": {
            "decision": true,
            "validators": null,
            "name": "between",
            "description": "Check to see if the trigger falls between two times,
and optionally on specific days.",
            "arguments": [
                {
                    "name": "from"
,                    "validator": {
                        "regex": "^[0-9]{2}:[0-9]{2}:[0-9]{2}$"
                    },
                    "description": "The 'from' time in hh:mm:ss 24hr format",
                    "type": "string",
                    "required": true
                },
                {
                    "name": "to",
                    "validator": {
                        "regex": "^[0-9]{2}:[0-9]{2}:[0-9]{2}$"
                    },
                    "description": "The 'to' time in hh:mm:ss 24hr format",
                    "type": "string",
                    "required": true
                },
                {
                    "name": "days",
                    "description": "The optional list of days in short form
(Mon,Tue,Wed...), for all days use a blank list []",
                    "type": "object",
                    "required": true
                }
            ],
            "actionOnAssociated": false,
            "type": ["alert","situation"]
        },
    "moolet_type": "alert"
}]
```

getWorkflows

A MoogDb v2 method that returns workflows for a Workflow Engine Moolet.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`getWorkflows`** takes the following request arguments:

Name	Type	Required	Description
`mooletName`	String	Yes	Name of the Workflow Engine Moolet to return workflows for.
`activeOnly`	Boolean		Return only active workflows.

Response

Method **`getWorkflows`** returns a JSON array of workflow objects. Each object has the following:

Name	Type	Description
`id`	Integer	Unique ID of the workflow.
`moolet_name`	String	Name of the Workflow Engine Moolet.
`workflow_name`	String	Name of the workflow.
`description`	String	Description of the workflow.
`sequence`	Integer	Sequence number of the workflow.
`active`	Boolean	Indicates whether or not the Moolet's associated Workflow Engine is active.
`entry_filter`	String	An SQL-like filter to determine which events, alerts, or Situations can enter the workflow. If empty, the workflow accepts all events, alerts or Situations.
`sweep_up_filter`	String	An SQL-like filter to intake any additional alerts or Situations from the database. Not relevant for event workflows.
`first_match_only`	Boolean	If enabled, alerts and Situations only pass through actions on the first time they enter the Workflow Engine. Not relevant for event workflows.
`operations`	JSON List	List of properties relating to each operation: Name / Type / Description type / String / Type of operation. Options are: 'action', 'decision' and 'delay'. operation_name / String / Name of the operation. Only relevant for 'action' and 'decision' types. function_name / String / Name of the function. Only relevant for 'action' and 'decision' types. function_args / JSON Object / Arguments for the function. duration / Integer / Length of time before the message goes to the next operation. Only relevant for 'delay' type. reset / Boolean / Determines whether the timer resets

		after each occurrence. Only relevant for 'delay' type.

Examples

The following examples demonstrate typical use of method **`getWorkflows`**:

Request example

Example request to workflows associated with the "Situation Workflows" Moolet:

```
var response = moogdb.getWorkflows("Situation Workflows");
```

Response example

Example response returning details of the workflows associated with the "Situation Workflows" Moolet:

```
[
  {
    "first_match_only": true,
    "sequence": 1,
    "operations": [
      {
        "duration": 120,
        "reset": false,
        "type": "delay"
      },
      {
        "operation_name": "Create Ticket",
        "function_name": "createServiceTicket",
        "forwarding_behavior": "always forward",
        "function_args": {
          "services": "ServiceNow, Remedy, Cherwell, Jira Service Desk, Jira
Software"
        },
        "type": "action"
      }
    ],
    "moolet_name": "Situation Workflows",
    "workflow_name": "Automated Ticketing",
    "entry_filter": "((category = \"Closed\") AND (custom_info.test = \"test\"))
AND (description = \"test\")",
    "active": false,
    "description": "You can optionally use this workflow if you use UI ticketing
integrations. It creates tickets in the ticketing system as Situations are
actioned.",
    "sweep_up_filter": "((sig_id = 1) AND (first_event_time = 1574121600)) AND
(description = \"test\")",
    "id": 1
  },
  {
    "first_match_only": false,
    "sequence": 2,
    "operations": [
      {
        "duration": 0,
        "reset": false,
        "type": "delay"
```

```
      },
      {
        "operation_name": "Stop Situation",
        "function_name": "stop",
        "forwarding_behavior": "Stop All Workflows",
        "type": "action"
      }
    ],
    "moolet_name": "Situation Workflows",
    "workflow_name": "Closed Situation Filter",
    "entry_filter": "status = 9",
    "active": true,
    "description": "You can optionally use this workflow to prevent closed
Situations from processing.",
    "sweep_up_filter": "",
    "id": 4
  }
]
```

mergeSituations

A MoogDb v2 method that merges two or more Situations, superseding the original Situations if required, and returns details of the newly created Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`mergeSituations`** takes the following request arguments:

Name	Type	Required	Description
`situationIds`	Native Array	Yes	A JSON array containing the IDs of the Situations to merge.
`keepOriginals`	Boolean	Yes	Determines what to do with the original Situations: **`true`**: Keep the original Situations. **`false`**: Supersede the original Situations.

Response

Method **`mergeSituations`** returns the following response:

Type	Description
CEvent	A CEvent object containing details of the newly created Situation.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

moveSituationToCategory

A MoogDb v2 method that moves a Situation into a new category. A category represents a type of Situation, indicating how it was created or its state. See Create Shared Alert and Situation Filters for more information.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`moveSituationToCategory`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.
`category`	String	Yes	Name of the new category you want to assign the Situation to.

Response

Method **`moveSituationToCategory`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`moveSituationToCategory`**:

Request example

Example request to move Situation ID 123 to the category `"Detected"`:

```
var result=moogdb.moveSituationToCategory(123, "Detected");
```

Response example

A successful request returns **`true`**.

moveSituationToQueue

A MoogDb v2 method that assigns a Situation to a queue and writes a thread entry if required. The queue and user may be provided as either an ID or a valid name.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **moveSituationToQueue** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	Situation ID.
user	Object	Yes	An object containing either a valid user name or ID.
queue	Object	Yes	An object containing either a valid queue name or ID
journal	String	No	An entry to add to the journal thread, if required.

Response

Method **moveSituationToQueue** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **moveSituationToQueue**:

Response example

A successful request returns **true**.

rateSituation

A MoogDb v2 method that applies a rating to a Situation.

Request arguments

Method **rateSituation** takes the following request arguments.

Name	Type	Required	Description
situationId	Number	Yes	ID of the Situation you want to rate.
rating	Number	Yes	Rating that you want to apply to the Situation. This is equivalent to the number of stars that you can assign to a Situation in the UI. One of:0 = Not yet rated1 = Bad2 = Poor3 = Adequate4 = Good5 = Excellent

comment	String	No	A comment about the rating you are applying to the Situation.

Response

Method **rateSituation** returns the following response:

Type	Description
Object	A Javascript object containing **rating**, **comment**, and **sig_id** of the rated Situation.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **rateSituation**:

Request example

Example request to apply a rating of 4 to Situation ID 18 with a comment "Rating 4":

```
var success = moogdb.rateSituation(18, 4, "Rating 4");
```

Response example

Example response returning the rating number, comment and ID of the rated Situation:

```
{"rating":4,"comment":"Rating 4","sig_id":18}
```

reload

A MoogDb v2 method that takes a Situation (CMooBotSituation) or alert (CMooBotAlert) type of CEvent and refreshes the data in the payload but preserves the metadata. This method should be used instead of getSituation and getAlert if you want to update the event with the latest data from the database, and when you are forwarding an event on using **situation.forward(<event>)**.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **reload** takes the following request arguments:

Name	Type	Required	Description
cevent	CEvent	Yes	A CEvent object representing the alert, containing alert attributes, such as type or severity.

removeAlertFromSituation

A MoogDb v2 method that removes an alert from a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`removeAlertFromSituation`** takes the following request arguments:

Name	Type	Required	Description
`alertId`	Number	Yes	Alert ID.
`situationId`	Number	Yes	Situation ID.

Response

Method **`removeAlertFromSituation`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`removeAlertFromSituation`**:

Response example

A successful request returns **`true`**.

removeSigCorrelationInfo

A MoogDb v2 method that removes all correlation information related to a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`removeSigCorrelationInfo`** takes the following request arguments:

Name	Type	Required	Description
`sitn_id`	Number	Yes	Situation ID.
`serviceName`	String	No	Service name.
`externalId`	String	No	External ID.

Response

Method **removeSigCorrelationInfo** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **removeSigCorrelationInfo**:

Response example

A successful request returns **true**.

removeSituationPrimaryTeam

A MoogDb v2 method that removes the primary team from a Situation. The team remains assigned to the Situation.

Request arguments

Method **removeSituationPrimaryTeam** takes the following request arguments:

Name	Type	Required	Description
sitn_id	Number	Yes	ID of the Situation that you want to remove the primary team from.

Response

Method **removeSituationPrimaryTeam** returns the following response:

Type	Description
Object	A Javascript object containing the Situation ID.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation

Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`removeSituationPrimaryTeam`**:

Request example

Example request to remove the primary team from Situation 1906:

```
var actions = moogdb.removeSituationPrimaryTeam(1906);
```

Response example

Example response returning the Situation ID that the primary team has been removed from:

```
{"sitn_id": 1906}
```

reorderWorkflows

A MoogDb v2 method that reorders the sequence of workflows within a Workflow Engine Moolet.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`reorderWorkflows`** takes the following request arguments:

Name	Type	Required	Description
`moolet_name`	String	Yes	Name of the Workflow Engine Moolet.
`workflow_IDs_sequence`	Array of Integers	Yes	An ordered array of all the workflow IDs within the Workflow Engine Moolet. The position of each workflow ID is its position within the Workflow Engine Moolet.

Response

Method **`reorderWorkflows`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

Examples

The following examples demonstrate typical use of method **`reorderWorkflows`**:

Request example

Example request to reorder the workflows in "Alerts Workflows" into the workflow sequence 1, 4, 3, 2, 5:

```
moogdb.reorderWorkflows("Alerts Workflows", [1, 4, 3, 2, 5]);
```

Response example

A successful request returns **`true`**.

resolveSituation

A MoogDb v2 method that resolves a Situation that is currently open.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`resolveSituation`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.

Response

Method **`resolveSituation`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`resolveSituation`**:

Response example

A successful request returns **`true`**.

reviveSituation

A MoogDb v2 method that revives (sets to Open) a Situation that is currently set to Resolved.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`reviveSituation`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.

Response

Method **reviveSituation** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **reviveSituation**:

Response example

A successful request returns **true**.

sendToWorkflow

A MoogDb v2 method that sends a Moolet Inform message to a workflow in an Inform Workflow Engine.

See Workflow Engine for more information on Inform Workflow Engines.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **sendToWorkflow** takes the following request arguments:

Name	Type	Required	Description
engineName	String	Yes	Name of an active Inform Workflow Engine.
workflowName	String	Yes	Name of an active workflow within the specified Inform Workflow Engine.
situationId	Number	No	ID of the Situation you want to send to the workflow.
situation	Object	No	Situation object to send to the workflow.
alertId	Number	No	ID of the alert you want to send to the workflow.
alert	Object	No	Alert object to send to the workflow.
context	String	No	Additional context to send with the message. This must be available as an action in the workflow as **getWorkflowContext()**.

Response

Method **sendToWorkflow** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

Examples

The following examples demonstrate typical use of method **sendToWorkflow**:

Request example

Example request to send a message to an Inform Workflow Engine:

```
var engineName = "Alert Inform Engine";
var workflowName = "My Alert Workflow ";
var context = { "key" : "value" , "key1" : "value" };
var sent = moogdb.sendToWorkflow(engineName,workflowName,alert,context);
```

Response example

A successful request returns **true**.

setAlertCustomInfo

A MoogDb v2 method that updates the custom information for an alert.

You can use this method either with the **alertInfo** CEvent or with both the **alertID** and **customInfoMap** arguments.

You can use the **merge** parameter alongside either method. This determines whether to merge the new custom information data with existing data or replace it.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **setAlertCustomInfo** takes the following request arguments:

Name	Type	Required	Description
alertId	Number	No, if you use **alertInfo**.	Alert ID. This can be used alongside **customInfoMap** and **merge** but not **alertInfo**.
alertInfo	CEvent	No, if you use **alertId**.	A CEvent containing **alert_id** and **custom_info** attributes, the values of which will be used to replace the **custom_info** in the specified alert.
customInfoMap	Object	Yes, if you use **alertId**.	A map of name value pairs containing the new **custom_info** information.
merge	Boolean	No	Determines the action for the custom information:**true**: Merge the existing data with the new data. Default.**false**: Replace the existing data with the new data.

Response

Method **setAlertCustomInfo** returns the following response:

Type	Description

Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`setAlertCustomInfo`**:

Response example

A successful request returns **`true`**.

setAlertSeverity

A MoogDb v2 method that sets the severity level for an alert.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`setAlertSeverity`** takes the following request arguments:

Name	Type	Required	Description
`alertId`	Number	Yes	Alert ID.
`severity`	Number	Yes	The severity of the alert as an integer:0 = Clear1 = Indeterminate2 = Warning3 = Minor4 = Major5 = Critical

Response

Method **`setAlertSeverity`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	No
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`setAlertSeverity`**:

Response example

A successful request returns **`true`**.

setPrcLabels

A MoogDb v2 method that sets the Probable Root Cause (PRC) labels for alerts within a Situation.

You can mark alerts as causal, non-causal or unlabeled within a Situation. An alert can have different PRC levels within different Situations.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`setPrcLabels`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.
`object`	JSON Object	Yes	A JSON object containing the following fields: **`causal`**: A JSON array containing the alert IDs that you want to be marked as causal. Leave the list empty if there are no causal alerts. **`non_causal`**: A JSON array containing the alert IDs that you want to be marked as non-causal. Leave the list empty if there are no non-causal alerts. **`unlabelled`**: A JSON array containing the alert IDs that you do not want to be labeled as causal or non-causal. Leave the list empty if there are no unlabeled alerts.

Response

Method **`setPrcLabels`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes

Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`setPrcLabels`**:

Request example

Example request to set alert IDs 1 and 2 as causal, alert IDs 3 and 7 as non-causal, and alert IDs 5 and 9 as unlabeled, in Situation ID 1:

```
var labelsSet = moogdb.setPrcLabels(1, { "causal" : [ 1, 2 ], "non_causal" : [
3,7 ], "unlabelled" : [ 5, 9] });
```

Response example

A successful request returns **`true`**.

setResolvingThreadEntry

A MoogDb v2 method that sets or clears a thread entry in a Situation as a resolving step. Threads are comments or 'story activity' on Situations.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`setResolvingThreadEntry`** takes the following request arguments:

Name	Type	Required	Description
`entryId`	Number	Yes	ID of the thread entry.
`resolving_step`	Boolean	Yes	Whether you are setting or clearing the thread entry as a resolving step.
`userId`	Number	Yes	A valid user ID.

Response

Method **`setResolvingThreadEntry`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **setResolvingThreadEntry**:

Request example

Example request to mark thread entry 32 as a resolving step using user ID 1:

```
var success = moogdb.setResolvingThreadEntry(32, true, 1);
```

Response example

A successful request returns **true**.

setSigCustomInfo

A MoogDb v2 method that updates the custom information for a Situation.

The Situation ID and new custom information are both contained in the **situationInfo** CEvent. The new custom information is contained in the **customInfoMap** object.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **setSigCustomInfo** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	Situation ID.
customInfoMap	Object	Yes	A map of name value pairs containing the new custom information.
merge	Boolean	No	Determines the action for the custom information:**true**: Merge the existing data with the new data. Default.**false**: Replace the existing data with the new data.

Response

Method **setSigCustomInfo** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`setSigCustomInfo`**:

Response example

A successful request returns **`true`**.

setSituationFlags

A MoogDb v2 method that updates the flags associated with a Situation. You can add flags to or remove them from a Situation.

See Situation Flags for more information on Cisco Crosswork Situation Manager Situation flags.

Request arguments

Method **`setSituationFlags`** takes the following request arguments:

Name	Type	Required	Description
`sitn_ids`	Array of Numbers	Yes	An array of IDs for the Situations you want to update.
`to_add`	Array of Strings	Yes	Flags to be added to those Situations. If this is an empty list, no flags are added to the Situation.
`to_remove`	Array of Strings	Yes	Flags you want to remove from the Situation. If this is an empty list, no flags are removed from the Situation.

Response

Method **`setSituationFlags`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`setSituationFlags`**:

Request example

Example request to update Situation IDs 1 and 2:

```
var result = JSON.stringify(moogdb.setSituationFlags([1, 2], ["S1","S2"],
["S2"]))
```

Response example

A successful request returns **`true`**.

setSituationPrimaryTeam

A MoogDb v2 method that sets one of the teams already assigned to a Situation as the primary team.

Request arguments

Method **`setSituationPrimaryTeam`** takes the following request arguments:

Name	Type	Required	Description
`sitn_id`	Number	Yes	ID of the Situation.
`team_id`	Number	No, if you specify **`team_name`**.	ID of the team that you want to make the primary team.
`team_name`	String	No, if you specify **`team_id`**.	Name of the team that you want to make the primary team.

Response

Method **`setSituationPrimaryTeam`** returns the following response:

Type	Description
Object	A Javascript object containing the Situation ID and the primary team ID.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`setSituationPrimaryTeam`**:

Request example

Example request to set the team "Database Management System" as the primary team on Situation 1906:

```
var actions = moogdb.setSituationPrimaryTeam(1906, 12);
```

Response example

Example response returning that team 12 is the primary team on Situation 1906:

```
{
    "primary_team_name": "Infrastructure",
    "sitn_id": 1906,
    "primary_team_id": 12
}
```

setSituationProcesses

A MoogDb v2 method that applies a list of processes to a Situation. Any other processes already associated with the Situation are removed.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`setSituationProcesses`** takes the following request arguments:

Name	Type	Required	Description
`situationId`	Number	Yes	Situation ID.
`processes`	JSON Array	Yes	A Javascript array of process names as text strings. If any processes supplied do not exist in the database, the request creates them and assigns them to the Situation.

Response

Method **`setSituationProcesses`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`setSituationProcesses`**:

Response example

A successful request returns **`true`**.

setSituationServices

A MoogDb v2 method that applies a list of external services to a Situation. Any other services already associated with the Situation are removed.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **setSituationServices** takes the following request arguments:

Name	Type	Required	Description
situationId	Number	Yes	Situation ID.
services	JSON Array	Yes	A Javascript array of service names as text strings. If any services supplied do not exist in the database, the request creates them and assigns them to the Situation.

Response

Method **setSituationServices** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

shareToolAccess

A MoogDb v2 method that shares access to a tool with other users, teams, or roles, or makes it global so that all users can access it. When a user creates a tool, it is automatically shared globally. You can use this endpoint to restrict its availability and ensure that tools are only available to users who need them. Using this endpoint to share access to a tool overwrites any existing shares.

Request arguments

Method **shareToolAccess** takes the following request arguments:

Name	Type	Required	Description
tool_id	Number	Yes	ID of the tool that you want to share access for.
domain	String	Yes	Domain to share access with. One of: **user**, **team**, **role**, or **global**.
domain_ids	Array	Yes/No	An array of one or more IDs within the domain. Optional for the **global** domain.

Response

Method **shareToolAccess** returns the following response:

Type	Description

Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

Examples

The following examples demonstrate typical use of method **shareToolAccess**:

Request example

Example request to share access of tool ID 15 with team ID 3:

```
var actions = moogdb.shareToolAccess(15, "team", 3);
```

Response example

A successful request returns **true**.

updateAlert

A MoogDb v2 method that takes an alert object and uses it to update the database and the Message Bus.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **updateAlert** takes the following request arguments:

Name	Type	Required	Description
alertObject	CEvent	Yes	Alert object containing the information about the alert.

Response

Method **updateAlert** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **updateAlert**:

Response example

A successful request returns **true**.

updateClosedAlert

A MoogDb v2 method that updates the description and custom info of a closed alert during the grace period. The grace period is when an alert is closed and in the active database, before it is archived to the historic database. If a custom info field already exists, this method replaces the previous value; if the custom info field does not exist, this method adds it.

The **`updateClosedAlert`** method returns an error if the alert is open, or if it is closed and has been archived to the historic database.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`updateClosedAlert`** takes the following request arguments:

Name	Type	Required	Description
`alert_id`	Number	Yes	ID of the closed alert that you want to update.
`description`	String	No	New description of the alert.
`custom_info`	JSON Object	No	A JSON object containing the custom info values that you want to update. If the key already exists, the method replaces the existing value. If the key does not exist, the method adds it.

Response

Method **`updateClosedAlert`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	No
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **`updateClosedAlert`**:

Request example

Example request to update the description and custom info for alert ID 9:

```
function updateClosedAlert1()
{
```

```
    var obtainAlert=moogdb.getAlert(9);
    var customInfo={"key1":"value1"};
    obtainAlert.set("description", "updateClosedAlert1_description_updated");
    obtainAlert.set("custom_info", customInfo);
    var result=moogdb.updateClosedAlert(obtainAlert);
    logger.warning("system updates alert which is in grace period, result should
return true");
    logger.warning("result = "+ result);
}
```

Response example

A successful request returns **`true`**.

updateClosedSituation

A MoogDb v2 method that updates the description and custom info of a closed Situation during the grace period. The grace period is when a Situation is closed and in the active database, before it is archived to the historic database. If a custom info field already exists, this method replaces the previous value; if a custom info field does not exist, this method adds it.

The **`updateClosedSituation`** method returns an error if the Situation is open, or if it is closed and has been archived to the historic database.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`updateClosedSituation`** takes the following request arguments:

Name	Type	Required	Description
`sitn_id`	Number	Yes	ID of the closed Situation that you want to update.
description	String	No	New description of the Situation.
custom_info	JSON Object	No	A JSON object containing the custom info values that you want to update. If the key already exists, the method replaces the existing value. If the key does not exist, the method adds it.

Response

Method **`updateClosedSituation`** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **`true`** = success, **`false`** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	No
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **updateClosedSituation**:

Request example

Example request to update the description and custom info for Situation ID 333:

```
function updateClosedSituation1()
{
    var obtainSit=moogdb.getSituation(333);
    var customInfo={"key1":"value1"};
    obtainSit.set("description", "updateClosedSituation1_description_updated");
    obtainSit.set("custom_info", customInfo);
    var result=moogdb.updateClosedSituation(obtainSit);
    logger.warning("system updates Situation which is in grace period, result
should return true");
    logger.warning("result = "+ result);
}
```

Response example

A successful request returns **true**.

updateCustomInfo

A MoogDb v2 method that updates the custom info for an alert or a Situation.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **updateCustomInfo** takes the following request arguments:

Name	Type	Required	Description
toUpdate	CEvent	Yes	A CEvent representing the alert or Situation you want to update.
toMerge	JSON Object	Yes	Custom info to add to or replace the existing custom info field.
merge	Boolean	No	Determines the action for the custom information:**true**: Merge the existing data with the new data. Default.**false**: Replace the existing data with the new data.

For an alert you can also use the following arguments:

Name	Type	Required	Description
alertId	Number	No	ID of the alert you want to add custom info to.
path	String	No	Dot-notation path to the **custom_info** key where the info is stored. Updates the existing value if the key already exists; creates the full path if the key does not exist.
param	Value	No	Value to put at the specified key.

Response

Method **updateCustomInfo** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

Examples

The following examples demonstrate typical use of method **updateCustomInfo**:

Response example

A successful request returns **true**.

updateMaintenanceWindow

A MoogDb v2 method that updates an existing maintenance window object, by passing an object containing the maintenance window information.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **updateMaintenanceWindow** takes the following request arguments:

Name	Type	Required	Description
updatedWindow	JSON Object	Yes	Maintenance window object containing the updated details.

The maintenance window object **updatedWindow** contains the following information:

Name	Type	Required	Description
id	Number	Yes	ID of the maintenance window you want to update.
user_id	Number	Yes	ID of the user who is updating the maintenance window.
name	String	No	Name of the maintenance window.
description	String	No	Description of the maintenance window.
filter	String	No	SQL-like filter that alerts must match to be included in the maintenance window.
start_date_time	Number	No	Start time of the maintenance window. This must be in Unix epoch time in seconds and may be up to 5 years in the future.
duration	Number	No	Duration of the maintenance window in seconds. The minimum duration is 1 second and the maximum is 157784630 seconds (5 years).
forward_alerts	Boolean	No	Determines whether or not alerts should be forwarded to the next Moolet in the processing chain.
recurring_period	Number	No	Whether or not this is a recurring maintenance window. Set this to:**1** for a recurring maintenance window.**0** for a one-time maintenance window.If not specified, default is **0**. If you set this property to **1**, you must specify **recurring_period_units**.

recurring_period_units	Number	No	Specifies the recurring period of the maintenance window, in days, weeks or months. Valid values are:**2** = daily**3** = weekly**4** = monthlyDefault is **0** if **recurring_period** is set to **0**.

Response

Method **updateMaintenanceWindow** returns the following response:

Type	Description
Object	A JSON object containing details of the updated maintenance window.

Examples

The following examples demonstrate typical use of method **updateMaintenanceWindow**:

Request example

Example request to update the description in maintenance window ID 2 by user ID 3:

```
var response = moogdb.updateMaintenanceWindow({ "id" : 2, "description" :
"Updated name", "user_id": 3 });
```

Response example

Example response returning details of the updated maintenance window ID 2:

```
{
  "del_flag": false,
  "forward_alerts": false,
  "last_updated": 1574164759,
  "timezone": "Europe/London",
  "description": "Updated name",
  "recurring_period_units": 0,
  "filter": "(severity IN (0, 1, 2, 3, 4, 5)) AND (owner IN (3))",
  "duration": 3600,
  "recurring_period": 0,
  "name": "Test",
  "updated_by": 3,
  "id": 2,
  "start_date_time": 1574164339
}
```

updateSituation

A MoogDb v2 method that takes a Situation object and uses it to update the database and the Message Bus.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **updateSituation** takes the following request arguments:

Name	Type	Required	Description
situationObject	CEvent	Yes	Object containing the Situation details.

Response

Method **updateSituation** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

API update behavior

The behavior of this method depends on whether the relevant alert or Situation is open, closed and still in the active database, or closed and archived to the historic database. This method updates or returns information about the alert or Situation as follows:

Alert/Situation Status	API Updates or Retrieves Alert/Situation
Open alert/Situation	Yes
Closed alert/Situation in active database	Yes
Closed alert/Situation in historic database	No

See API Update Behavior for more information on Situation statuses.

Examples

The following examples demonstrate typical use of method **updateSituation**:

Response example

A successful request returns **true**.

updateTeam

A MoogDb v2 method that updates an existing team, by passing an object containing team information.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **updateTeam** takes the following request arguments:

Name	Type	Required	Description
teamObj	Object	Yes	Object containing the team information.

The team object **teamObj** contains the following information:

Name	Type	Required	Description
team_id	Number	Yes	Team ID.
name	String	No	Team name. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
alert_filter	JSON Object	No	An SQL-like or JSON filter that alerts must match to be assigned to the team. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
services	Array of Numbers or Strings	No	List of the team service names or IDs. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
sig_filter	JSON Object	No	An SQL-like or JSON filter that Situations must match to be assigned to the team. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.

landing_page	String	No	Default landing page for the team. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
active	Boolean	No	Set to **true** if the team is active; set to **false** if the team is inactive. Default is **true**. Exclude this attribute to leave Cisco Crosswork Situation Manager as is.
description	String	No	Team description. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.
users	Array of Numbers or Strings	No	List of users in the team, either IDs or usernames. Exclude this attribute to leave Cisco Crosswork Situation Manager as it is.

Response

Method **updateTeam** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

Examples

The following examples demonstrate typical use of method **updateTeam**:

Response example

A successful request returns **true**.

updateUser

A MoogDb v2 method that updates an existing user, by passing an object containing user information.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **updateUser** takes the following request arguments:

Name	Type	Required	Description
userObj	Object	Yes	Object containing the user details.

The user object **userObj** contains the following information:

Name	Type	Required	Description
username	String	No, if you use **uid**.	Username of the user to be updated.
uid	Number	No, if you use **username**.	User ID of the user to be updated.
password	String	No	New user password, only valid for DB realm.
active	Boolean	No	Set to **true** if the user is active, **false** if the user is inactive. Default is **true**.

email	String	No	User's email address.
fullname	String	No	User's full name.
roles	JSON Array	No	List of either the role IDs or the role names. For example, **"roles":["Super User"]**.
primary_group	String or Number	No	User's primary group name or primary group ID.
department	String or Number	No	User's department ID or department name.
timezone	String	No	User's timezone.
contact_num	String	No	User's phone number.
session_expiry	Number	No	Number of minutes after which the user's session expires. Default is the system default.
competencies	JSON Array	No	A list with the user competencies. Each competency should have have name or cid and ranking. For example: **[** **{"name":"SunOS", "ranking": 40},** **{"name":"SAP", "ranking": 50},** **{"name":"EMC", "ranking": 60}** **]**
team	JSON Array of Numbers or Strings	No	List of the user's team names or team IDs.

Response

Method **updateUser** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

Examples

The following examples demonstrate typical use of method **updateUser**:

Request example

Example request to update a number of fields in user ID 5:

```
{
    "uid": 5,
    "fullname": "Phil Customer",
    "competencies": [
    {
        "name": "SunOS",
        "ranking": 40
    },
    {
        "name": "SAP",
        "ranking": 50
    },
    {
```

```
            "name": "EMC",
            "ranking": 60
        }],
        "roles": ["Super User"],
        "department": 3,
        "active": true,
        "email": "phil@example.com",
        "timezone": "(GMT 00:00) Europe/Jersey - Greenwich Mean Time",
        "teams": [1, 2, 4],
        "joined": 12345678,
        "contact_num": "0965412345"
    }
```

Response example

A successful request returns **`true`**.

updateWorkflow

A MoogDb v2 method that updates an existing workflow in the Workflow Engine.

Back to MoogDb V2 API Method Reference.

Request arguments

Method **`updateWorkflow`** takes the following request arguments:

Name	Type	Required	Description
`id`	Integer	Yes	ID of the workflow you want to update.
`details`	JSON Object	Yes	A JSON object containing the details of the workflow(s) to be updated.

The object **`details`** contains the following information:

Name	Type	Required	Description
`workflow_name`	String	Yes	Name of the workflow.
`active`	Boolean	No	Determines whether the workflow is active or not. If **`true`**, the workflow is active.
`description`	String	No	Description of the workflow.
`entry_filter`	String	No	An SQL-like filter to determine which events, alerts or Situations can enter the workflow. If empty, the workflow accepts all events, alerts or Situations.
`sweep_up_filter`	String	No	An SQL-like filter to intake any additional events, alerts or Situations from the database.
`first_match_only`	Boolean	No	Determines whether to perform workflow operations only once on each object.
`operation`	JSON Array	No	List of properties relating to each operation:

			Name	Type	Required	Description
			type	String	Yes	Type of operation. Options are: 'action', 'decision' and 'delay'.
			operation_name	String	Yes, for 'action' and 'decision' types.	Name of the operation.
			function_name	String	Yes, for 'action' and 'decision' types.	Name of the function.
			forwarding_behavior	String	No	Forwarding behavior for the function. One of:**`always forward`**: The function always forwards the object to the next workflow.**`stop this workflow`**: The function stops this workflow and the object moves to the next workflow.**`stop all workflows`**: The function stops all workflows for this object.Default is **`always forward`**. Only valid for 'action' and 'decision' types.

			function_args	JSON Object	No	Arguments for the function.
			duration	Integer	Yes, for 'delay' type.	Length of time before the message goes to the next operation.
			reset	Boolean	Yes, for 'delay' type.	Determines whether the timer resets after each occurrence.
reset	Boolean	No	Mandatory for 'delay' type.			

Response

Method **updateWorkflow** returns the following response:

Type	Description
Boolean	Indicates whether or not the operation was successful: **true** = success, **false** = fail.

Examples

The following examples demonstrate typical use of method **updateWorkflow**:

Request example

Example request to update workflow ID 1 with a new workflow name:

```
moogdb.updateWorkflow(1, {workflow_name: "new name"});
```

Response example

A successful request returns **true**.

LAMbots

Lambot Overview

LAMbot Configuration

LAMbots are JavaScript modules associated with every LAM. The LAMbots control the actions the LAM performs at startup and any necessary processing before forwarding objects to the Message Bus.

You can configure a LAMbot by modifying the functions and modules within its configuration file. The LAMbot files are located at **$MOOGSOFT_HOME/bots/lambots.**

LAMbot Functions

Each LAMbot includes an **onLoad** function that runs at startup and a **presend** function that processes and filters objects before sending them to the Message Bus.

REST-based LAMbotscall **modifyResponse** after they receive an object and convert it to JSON.

REST Client-based LAMbots call **preClientSend** before they send a request to a polled server and **modifyResponse** after a response is received from a polled server.

onLoad

Every instance of a LAMbot calls the **onLoad** function at startup. We recommend setting up shared values or lookup tables in the **onLoad** function. You can use it to initalize internal variables, load external JavaScript modules and set up structures needed for the filter function. For example:

```
var config = MooBot.loadModule('Config');
var moogUrl;

function onLoad()
{
    var servletsConf = config.getConfig('servlets.conf');
        if (servletsConf)
        {
        moogUrl = servletsConf.webhost;
        }
}
```

The **onLoad** function:

- Stores the value of the servlets configuration in the **config** module to a variable "servletsConf".
- Sets the variable **moogURL** to the servlets **webhost** value.

presend

The LAMbot calls the **presend** function every time it assembles an object to publish on the Message Bus. Moogfarmd processes objects and turns them into alerts and Situations. An example **presend** function is:

```
function presend(event)
{
        event.setCustomInfoValue("eventDetails",overflow);
        if (overflow.LamInstanceName && (overflow.LamInstanceName ===
"DATA_SOURCE"))
        {
        delete overflow.LamInstanceName;
    }
        event.setCustomInfoValue("nodeSeverity", overflow.Severity);
    event.setCustomInfoValue("nodeMachineType", overflow.MachineType);
    event.setCustomInfoValue("nodeVendor", overflow.Vendor);
    return true;
}
```

The **presend** function:

- Adds the **overflow** object as event details.
- Checks whether LamInstanceName is the default value DATA_SOURCE and if so, removes it from custom info.
- Saves three overflow fields to custom info.

- Returns a true response to indicate that the object will be passed to the Message Bus.

You can partition event streams into substreams for differential processing in a distributed environment. You can send a boolean response if the configuration dictates that all objects will or will not be sent to the bus.

Instead of a boolean response, you can configure the function to return a JSON object containing two members: "passed" which is either true or false, and "stream" which defines the substream to send the event. For example:

```
function presend(event)
{
    return
    ({
        "stream" : "my_stream",
        "passed" : true
    });
}
```

You can configure the event inside the **presend** function. For example you can:

- Change values
- Access lookup tables
- Add or remove key value bindings
- Access regular expressions
- Extract tokens

In the LAMbot, the following line instructs the LAM to use the **presend** function. It calls **filterFunction** using the global **LamBot** variable:

```
LamBot.filterFunction("presend");
```

The **filterFunction** function receives a string, which is the name of the function to use for filtering.

You define the **presend** processing file or stream in individual LAM configuration files. See "Filtering" in Data Parsing for more information.

preClientSend

REST Client-based LAMbots call **preClientSend** before they send a request to a polled server. The function accepts an object and returns a modified version that is then sent by the Rest Client LAM. An example **preClientSend** function is:

```
function preClientSend(outBoundEvent)
{
    outBoundEvent.set('method', 'Post');
    var header = outBoundEvent.value('header');
    header['Content-Type'] = 'application/json';
    outBoundEvent.set('header', header);
    var body = { 'events': 'all', 'type': { 'id': '12345', 'name': 'incident' }
};
    outBoundEvent.set('body', body);
    return true;
}
```

The function generates a POST request with body type JSON.

In the LAMbot, the following line instructs the LAM to use the **preClientSend** function. It calls **preClientSendFunction** using the global **LamBot** variable:

```
LamBot.preClientSendFunction("preClientSend");
```

modifyResponse

You can modify the response sent by a REST-based LAMbot after it receives an object and a REST Client-based LAMbot after it receives a response from a polled server. An example **modifyResponse** function is:

```
function modifyResponse(inBoundEventData)
{
    var response = JSON.parse(inBoundEventData.value('responseData'));
    if (inBoundEventData.value('moog_target_name') == 'target1') {
        response['manager'] = 'primary';
    }
    else {
        response['manager'] = 'secondary';
    }
    inBoundEventData.set('responseData', JSON.stringify(response));
    return true;
}
```

The function generates a different response depending on the name of the REST client target called.

In the LAMbot, the following line instructs the LAM to use the **modifyResponse** function. It calls **modifyResponseFunction** using the global **LamBot** variable:

```
LamBot.modifyResponseFunction("modifyResponse");
```

LAMbot Modules

You can load modules into a LAMbot to perform various tasks. The most commonly used modules are:

- Logger: Cisco Crosswork Situation Manager components generate log files to report their activity.
- Constants: Used to share logic, states and flags between LAMbots.
- Utilities: A JavaScript utility used to escape and convert XML strings and JSON objects.

Define a global object to load a module into a LAMbot. For example:

```
var logger = LamBot.loadModule("Logger");
var constants = LamBot.loadModule("Constants");
var utilities = LamBot.loadModule("Utilities");
```

Moobots

Moobot Modules

Within Cisco Crosswork Situation Manager data processing, Moogfarmd Moolets, LAMs and integrations use simple computer programs called "bots" to perform automated tasks. A Moobot is a JavaScript file that is loaded at startup by a Moolet. The Moobot exposes logic and data flow, which you can control in JavaScript, relevant to the necessary function. LAMbots perform a similar function for LAMs and integrations.

Moobots expose the function of the Moolets allowing for extensive customization, for example in the Alert Rules Engine where the Moobot is used to perform automation.

Threads and global scope

Cisco Crosswork Situation Manager is built to handle high scale environments, so individual JavaScript MooBots are run in a multi-threaded fashion. For example, if a Moolet has ten threads, there will be ten instances of the MooBot running. This supports high throughput of Events through the Moobot, particularly, when they are doing complex processing. However, it does have important implications for the JavaScript concerning where the global scope (or context) for the JavaScript program for the MooBot resides. In principle, each Moobot has its own independent global scope. So it is impossible for one Moobot's logic to interact and affect another instance of the Moobot logic. To allow necessary communication between individual Moobot instances there are utility modules such as the Constants module.

Moobot modules

You can use the available Moobot modules to perform these functions:

Module	Description
Config	Read configuration files within LAMbots and Moobots.
Constants	Build a key value dictionary shared across Moobots.
Events	Set the types of Event that interest a Moobot.
ExternalDb	Access external relational databases.
Graph Topology	Access topology methods.
Kafka	Allows you to broadcast information on a Kafka bus.
LoggerConfigure Logging	Write log messages to the common Moogfarmd log file. See Configure Logging.
Mailer	Send an email in response to events occurring in Cisco Crosswork Situation Manager.
MoogDb V2	Query and manipulate a variety of entities in the Cisco Crosswork Situation Manager database, including alerts and Situations.
Moolet Informs	Can send update messages from one Moolet to other Moolets.
Process	Run and control the execution of other processes.
RabbitMQ	Allows you to broadcast information on a RabbitMQ bus.
REST.V2	Access an external RESTful API via HTTP to post, read, or delete data.
Utilities	Escape and unescape XML strings, convert an XML string to a JSON object and vice versa.

To use these modules, define a global variable at the top of the Moobot js file using the **`loadModule`** method.

You can also load load external JavaScript modules using the **`loadModule`** method. See below.

Examples

Throughout this section, all examples will use **`AlertBuilder.js`** to explain how Moobots function.

Step 1

When the Alert Builder starts and creates an instance of the Moolet, it creates a Moobot for every threaded instance of the Moolet. The first action undertaken by a Moobot is to load a system wide default file called **`MooB.js`**. This file pushes into the Global Scope using a closure, some shared functionality, which you can take advantage of in the Moobot. You should never edit **`MooB.js`** as the file is linked to the internal implementation of the Moobots.

Step 2

The preload statements in the **`MooB.js`** closure instruct a Moobot to load into its Global Scope the available modules. For example, they can be used to:

- Change and create structure in the MoogDb database.
- Listen for specific events in the system.
- Push events out.
- Log to the common log file output.
- Communicate using communication methodologies such as tweets, email etc.

Before you can use any of the built in modules that correspond to the functionality Cisco provides, you need the **`preload()`** method in the global object (MooB.js) to load the required modules.

The object exposes an API that you can use to add functionality into the system. In the example above, "**`Process`**" has a number of functions that you can call which allow the Moolet to run processes in the system.

After loading and running the **`MooB.js`** closure in the Moobot, the full Moobot user definable JavaScript file is loaded and run. It is important to understand from a JavaScript concept that it is executed at start-up. The reason for executing the script at start-up is to load any Event driven callbacks, and initialization code inside of the Moobot. For example in the Alert Builder, for a new Event arriving in the Moolet, Cisco Crosswork Situation Manager needs to know which functionality inside of the Moobot to run.

Using external modules

Moobots can load external JavaScript modules. This means that modules can be reused as generic functions in multiple Moobots.

To do this:

- Add the external JavaScript module file (**`BotExampleModule.js`**) in the **`$MOOGSOFT_HOME/bots/moobots`** or the **`$MOOGSOFT_HOME/contrib`** directory.
- Load the external JavaScript module in the Moobot by adding a line at the beginning (relative paths are supported), for example:

```
MooBot.loadModule('BotExampleModule.js');
```

The example below shows the external JavaScript module (**`BotExampleModule.js`**). It defines a class which takes an alert and prints out a message:

```
function CPrinter()
{
```

```
    var mLogger=MooBot.loadModule('Logger');
    var self=
    {
        prettyPrint: function(alert)
        {
            mLogger.info("This is a print of " + alert.value("alert_id") + "
other info");
        }
    };
    var F=function() {};
    F.prototype=self;
    return( new F() );
}
```

The **`AlertMgr.js`** Moobot loads the external JavaScript module **`BotExampleModule.js`** and uses the function **`CPrinter`** (from the external JavaScript module) to send alert details to a remote service:

```
MooBot.loadModule('BotExampleModule.js');
var printer = new CPrinter();
function newAlert(alert)
{
    printer.prettyPrint(alert);
}
```

onLoad function

Moobots can include an **`onLoad`** function to allow commands to be run once on startup per Moobot instance. You can use it to initialize internal variables, such as **`dbTypes`**, for example:

```
var dbTypes = null;
function onLoad()
{
    dbTypes = {
        employees: {
            type: 'mySql',
            host: '192.168.1.141',
            port: '3306',
            database: 'emp_db'
        },
        customers: {
            type: 'sqlServer',
            host: '213.32.112.17',
            database: 'customers',
            user: 'sa',
            encrypted_password: '0rJGl5oCWpmE9Hbk32sxFgxlQV3O5cx2bx1vKNOM7YA='
        }
    };
}
```

Config

The Config bot module allows you to read configuration files within LAMbots and Moobots.

It retrieves valid JSON configuration files found in **`$MOOGSOFT_HOME/config`** and performs a direct read from the file system before delivering the JSON Object to the calling bot. The module is available for all bots but can only be used for reading and storing global configuration files.

Before you begin

Before you use the Config bot module, ensure you have met the following requirements:

- The configuration file is in valid JSON.
- The configuration file is in **`$MOOGSOFT_HOME/config`**.
- The configuration is present on the file system as the process running the bot.

Best practice

Follow these guidelines when using the Config bot module:

- Use the module within the constraints of the **`OnLoad`** function.
- Note that making multiple calls to the module may impact the performance of the bot.
- Keep custom configuration files in a subdirectory of **`$MOOGSOFT_HOME/config`** and name them appropriately.
- Comment custom configuration files extensively so other users can understand the context of their use.

Error reporting

The following error messages are returned if the configuration file cannot be opened, the contents returned are null or if the JSON is invalid:

```
INFO :[CJSONCodec.java]:813 +|java.io.FileNotFoundException:
/export/src/incident/build/config/bad.conf (No such file or directory): Unable
to open file /export/src/incident/build/config/bad.conf|+

WARN :[CJSONCodec.java]:105 +|Failed to parse file
/export/src/incident/build/config/bad.conf, returned null contents|+

WARN :[CConfigModule.java]:112 +|File
[/export/src/incident/build/config/bad.conf] is either missing, unreadable or is
not valid JSON.|+
```

Examples

If you want to create a URL that links to Cisco Crosswork Situation Manager Situations, you can use the Config bot module to dynamically retrieve the base URL of the Cisco Crosswork Situation Manager instance from **`servlets.conf`**. For example:

```
var config = MooBot.loadModule('Config');
...
var servletsConf = config.getConfig('servlets.conf');
if (servletsConf) {
    moogURL = servletsConf.webhost;
}
```

Constants

Each Moobot runs in its own thread and instances of Moobots are independent of each other. The Constants module enables you to share logic, states or flags between Moobots. You can build a key value dictionary mapping that is shared across Moobot instances.

There are many system wide defined Constants that are used in the Events module to define which event to listen for. See the event types table below for more information.

Load the module

The Constants module is available to load into any standard Moobot. To load it, define a new global variable at the top of the Moobot Javascript file. For example:

```
var constants = MooBot.loadModule('Constants');
```

Method reference

The Constants module uses the following methods.

put

Associated a specified value with a specified key. Replaces the mapping for an existing key.

Request arguments

The method takes the following arguments.

Name	Type	Description
key	String	The key to associate with the value.
value	Object	The value to associate with the key.

Response

None.

get

Retrieves the value mapped to a specified key.

Request arguments

The method takes the following arguments.

Name	Type	Description
key	String	The key for which to retrieve the value.

Response

The method returns the following parameter:

Type	Description
Object	The value to which the key is mapped, or null if no mapping exists.

contains

Returns a positive response if the module contains an object with the specified name.

Request argument

The method takes the following arguments:

Name	Type	Description
name	String	The Object name.

Response

The method returns the following parameter:

Type	Description
Boolean	True if the module contains the specified object, otherwise false.

remove

Removes the value mapped to the specified key.

Request arguments

The method takes the following arguments:

Name	Type	Description
key	String	The key for which the value is to be removed.

Response

None.

eventType

Retrieves the value of a specified event type.

Request arguments

The method takes the following arguments:

Name	Type	Description
name	String	The name of the event type. See the list of event types in the table below.

Event types

You can use one of the following event types in the name argument:

Name	Passed Value	Description
E_LamEvent	**"Event"/"Events"**	Raw event from a LAM
E_NewAlert	**"Alert"/"Alerts"**	New alert
E_AlertUpdate	**"AlertUpdate"**	Alert update
E_CloseAlert	**"AlertClose"**	Close alert
E_NewComment	**"Comment"**	New comment
E_NewFeedback	**"Feedback"**	New feedback
E_NewSig	**"Sig"**	New Situation
E_SigClose	**"SigClose"**	Close Situation
E_SigUpdate	**"SigUpdate"**	Updated Situation
E_SigStatus	**"SigStatus"**	Situation status
E_SigAction	**"SigAction"**	Situation action
E_ThreadEntry	**"ThreadEntry"**	A thread entry
E_NewThreadEntry	**"NewThreadEntry"**	A new thread entry

E_Summary	**"Summary"**	System summary
E_Invite	**"Invitation"**	Situation Room invitation
E_User	**"User"**	Username
E_Unknown	**"Unknown"**	An uncategorized event (error condition)

Response

The method returns the following parameter:

Type	Description
CEvent	An object containing the value of the specified event type.

Examples

The following example in **AlertBuilder.js** shows the Constants module with two methods that allow you to post and retrieve values from a shared scratchpad.

```
var count=0
constants.put("counter",count);
```

The variable count is set to 0 and stored using the label **counter**.

You can then retrieve a value by calling the **get** method and passing the name of the shared attribute, which is returned as a JavaScript local variable.

```
var count_val=constants.get("counter");
count_val++;
constants.put("counter",count_val);
```

- The **get** method takes the name of the shared attribute, "counter".
- The variable **count_val** is incremented.
- The **put** method takes the name of the variable to store, "counter", and the incremented value **count_val**.

If nothing is stored in **counter**, the Moobot returns **null**.

The following example passes the name of an event and returns a system wide constant that identifies that type of event when using the Events module.

```
constants.eventType("Event")
```

Events

The Events Moobot module allows you to make a Moobot driven by the occurrence of events by defining the type of event that interests the Moobot. It is available to load into any standard Moobot.

To use, at the top of a Moobot js file, define a new global variable **events** to load the Events module:

```
var events = MooBot.loadModule('Events');
```

Note

Compatibility with MoogDb and MoogDb V2 Methods and auxiliary objects listed here are compatible with the MoogDb V2 module.

Method

events.onEvent

The Events module has only one method, **onEvent**. This method points the Moobot to a supplied JavaScript function, which is called when a specified event type occurs.

The parameters to the called function depend on the type of event that you are listening for.

In a Moobot, this method is typically the last line in a script.

The type of event adaptor chosen is specific to the type of Moobot you are building.

onEvent method

Takes the name of a valid JavaScript function in a Moobot and also event code (from the Constants module **eventType**), and returns an event adaptor object

Request Arguments

Name	Type	Description
functionName	String	The name of a valid JavaScript function in the Moobot that is called when the event arrives.
type	CEvent	**eventType** event code that specifies what type of event the Moobot is listening for. It is typically from the Constants module.

Return Parameter

Name	Type	Description
CEventAdaptor	Object	An event adaptor object. Made active with the **listen ()** function in-line to listen for the event type.

Example

For the AlertBuilder MooBot:

```
events.onEvent("newEvent",constants.eventType("Event")).listen();
```

- Call the newEventJavaScript function.
- Define the Event type Event (from the Constants module), which responds to events put on the Message Bus by a LAM.
- Call the listenfunction in-line to listen for the event type.

When the Moolet starts and loads this events Moobot, its JavaScript file executes, initializing the Moobot to respond in an event-driven way to events arriving.

newEvent JavaScript function

The format of the function newEvent (which is called when you get an event), is as follows:

```
newEvent()
```

Request Argument

Name	Type	Description

event	CEvent object	An object that encapsulates all the data for the event from the Message Bus, and allows you to forward the event to the bus, using the CEvent forward method detailed below.

Event forward methods

The advantage of this approach is that alerts / Situations can be forwarded to different AlertRulesEngines / Sigalisers dynamically in the Moobots (for example based on the value of the source file).

```
alert.forward("Cookbook");
```

You could instead remove the **process_output_of** lines from the AlertRulesEngine / Sigaliser / Cookbook / Speedbird Moolets and explicitly send events / alerts / Situations on within the Moobot code using (as an example):

You can emulate MoogDb behavior by running the MoogDb.V2 Moobots.For example, the **alert.forward(this) line** will send an alert onto the Moolets specified in the appropriate **process_output_of** block within **moog_farmd.conf**.

CEventAdaptor auxiliary object

This object is a utility class used by the Events module to allow for the programmatic activation of event listening. It has one method:

```
listen()
```

Starts the event adaptor listening, which then calls the specified function when an event occurs.

Request Argument

None.

Return Parameter

Void - no value returned.

CEvent auxiliary object

This object encapsulates a generic Message Bus event object, and the contents of it are specific to the event type it represents. You can however access the key-value pairs contained in the object, and also set the values. Its methods include:

contains

Returns true if the Event contains a value stored at the key **name**.

Request Argument

Name	Type	Description
name	String	The name of the key being queried.

Return Parameter

Type	Description
Boolean	True if the event has a field called **name**, otherwise false.

set

Associates the specified **value** with the specified **name** in the event.

Previous **key** mapping has the old **value** replaced.

Request Argument

Name	Type	Description
name	String	The key with which the specified value is to be associated.
value	Object	The value associated with the key.

Return Parameter

Type	Description
Boolean	Indicates if the operation was successful: true = success, false = fail

value

Returns the object stored at the key **name**.

Request Argument

Name	Type	Description
name	String	The name of the key to return the object from.

Return Parameter

Type	Description
Object	A Javascript object containing what is at the key **name**.

CEvents API

The CEvents API is an object interface used to encapsulate data as it flows through Cisco Crosswork Situation Manager. A CEvent object contains status and data, and methods to access and manipulate that data. The data contained in the CEvent object depends on the type specified in the object, which include LAM events, alerts, Situations, thread entries, and invitations.

This API uses the following methods.

contains

Checks whether the CEvent object contains the given key.

Request arguments

Name	Type	Description
key	String	Name of a potential key in the CEvent object.

Return parameter

Type	Description
Boolean	Returns **true** if the provided key exists in the CEvent object, or **false** if it was not.

Request example

```
var custom_info = event.contains("custom_info") ? event.getCustomInfo() : {};
```

evaluateFilter

Allows an event/alert/Situation to be easily evaluated against a filter.

Request arguments

Name	Type	Description
filter	String	An SQL-like filter for events, alerts or Situations.

Return parameter

Type	Description
Boolean	Whether the filter matches the event, alert or Situation. Returns **true** if the filter matches the event, alert or Situation. Returns **false** if the filter has a correct syntax but doesn't match the event, alert or Situation. Returns **null** if the filter syntax is incorrect.

Request example

```
var is_matching = situation.evaluateFilter("description LIKE 'Created
Situation'");
```

forward(*this*)

Forwards the CEvent down the chain configured in the **moog_farmd.conf** (using the **process_output_of** configuration). The usual way of calling this is **CEvent.forward(this)** where **this** is the Moobot that is processing the CEvent object. This method also sends the CEvent object to any Moolet listening via event_handlers.

Request arguments

Name	Type	Description
moobot	NativeObject	The instance of the Moobot which is handling the CEvent object, usually the variable named **this**.

Return parameter

None.

forward(*target*,....)

Takes any number of target Moolet names as strings and forwards the CEvent to each of them. For example CEvent.forward("moolet1") or CEvent.forward("moolet1", "moolet2").

Request arguments

Name	Type	Description
targets	Stringvarargs	One or more Moolet names as strings.

Return parameter

None.

Request examples

You can forward alerts or Situations to other Moolets such as clustering algorithms programmatically using this function.

Example request to forward an alert to Alert Enricher:

```
alert.forward("AlertEnricher");
```

Example request to forward a Situation to Situation Manager Labeler:

```
situation.forward("SituationMgrLabeller");
```

getActionDetails

A utility helper method that retrieves the entire alert or Situation contained in the payload of a CEvent. The format of the details varies depending on what the action type is, and may be empty.

Request arguments

None

Return parameter

Type	Description
JS NativeObject	Whole of the alert or Situation contained in the payload of the CEvent, as a NativeObject ready for use in the Javascript for a Moobot.

getCorrelationInfo

Returns the correlation information for a Situation, which lists all of the services which are interested in this Situation. This method only applies to CEvent objects that contain Situation thread entries from the Collaborate tab in a Situation Room. For other correlation information, use the MoogDb v2 method getSigCorrelationInfo.

Request arguments

None

Return parameter

Type	Description
NativeObject	An object which contains the sig_id, service_name, external_id and properties for all the correlation info for the Situation. sig_correlation_info is a one to many relationship of sigs to services.

getCustomInfo

A helper method provided to retrieve the whole custom_info object for an alert or Situation.

Request arguments

None

Return parameter

Type	Description
JS NativeObject	Whole custom_info map for an alert or Situation as a NativeObject ready for use in the Javascript for a Moobot.

Bot.getType

Returns the internal name of the Moobot that is running the code.

Request arguments

None.

Return parameter

Type	Description
Enumerated type	Can be one of the following:

Request example

Example request if the following code is put into the Alert Builder Moobot:

```
logger.warning("This moobot is a: " + Bot.getType());
```

When Moogfarmd is started, the log line shows:

```
[AlertBuilder.js:65] +|This moobot is a: CAlertBuilder|+
```

getSummaryData

Returns a summary of information about a system, such as the number of alerts or the service count bundled up as key/value pairs.

Request arguments

None.

Return parameter

Type	Description
JS NativeObject	The summary of information about a system: summary.alert_count - number summary.service_count - number summary.sig_summaries - map (contains "categories" and "queues") summary.sig_summaries.categories - (array of objects) summary.sig_summaries.queues - (array of objects) Categories and queues contain the following: summary.sigs_down - number summary.sigs_up - number summary.total_events - number summary.total_sigs - number

Request example

If a Moolet is configured to listen to the 'Summary' event type as follows:

```
events.onEvent("summary", constants.eventType("Summary")).listen();
```

Then you can define a function can be defined to extract data out of the summary event object as follows:

```
function summary(summary)
{
var info = summary.getSummaryData();
logger.warning("Summary data: Events: "+info.total_events + " Situations: " +
info.open_sigs);
}
```

getTopic

Returns the topic that the data was received on, for example "alerts" or "Situations".

Request arguments

None.

Return parameter

Type	Description
String	Name of the topic that the data came from or relates to, such as "Situations" or "alerts".

payload

Retrieves the whole data payload that was sent in the CEvent object. In most cases the data contained in the payload is going to represent either a Situation or an alert, and as such will have key/value pairs which match the data columns for each.

Request arguments

None.

Return parameter

Type	Description
CMooMsg	Enum value specifying the type of data that the Event contains and/or which topic the data was received on from the bus.

Examples

Request example

Example CEvent payload request:

```
logger.warning(cevent.payload().getData());
```

Response example

Example CEvent payload response:

```
{active=true, competencies=[], contact_num=, department=null,
description=Online, email=, fullname=cyber, groupname=End-User, invitations=[],
joined=1516963803, only_ldap=0, photo=-1, primary_group=1, profile_image=null,
realms=[DB], roles=[1, 3, 4, 5], session_expiry=null, status=1, teams=[],
timezone=SYSTEM, uid=6, username=cyber}
```

set

Inserts or updates a value in the CEvent object. This call does no transformation of values. All values specified must match the underlying value type in the CEvent.The custom_info value is a JSON string. If

using **.set()** to change the value of custom_info, the JS object must be stringified first. Use **setCustomInfo()** to update custom_info.

Request arguments

Name	Type	Description
key	String	Key to insert or change a value at.
value	String or Number	New value to store against the key.

Return parameter

Type	Description
Boolean	Indicates whether or not the value was successfully changed: **true** = success, **false** = fail.

setCustomInfo

Sets the whole custom_info object for an alert or Situation.

Request arguments

Name	Type	Description
customInfo	NativeObject	The whole custom_info object to set for an alert or Situation.

Return parameter

None.

setCustomInfoValue

Sets a value of a specific property within custom_info to the supplied value. This can be used to change existing values, or create new ones.

Request arguments

Name	Type	Description
field	String	Dot-formatted field within the custom_info of the reference alert or Situation to update.
value	String, Integer, Boolean, Object, or Map	String, integer, Boolean, object, or map value to replace the value stored in the custom_info field.

Return parameter

Type	Description
Boolean	Indicates if the operation was successful: **true** = success, **false** = fail.

Request examples

You can use this method to add or replace specific keys within alert or Situation custom_info.

Example request to set a custom_info value in an alert:

```
alert.setCustomInfoValue("key1.my_new_key", "my_new_value");
var result = moogdb.updateAlert(alert);
```

Example request to set a custom_info value in a Situation:

```
situation.setCustomInfoValue("fieldA.fieldB",
{"my_new_map_key1":"my_new_map_value1"});
var result = moogdb.updateSituation(situation);
```

setTopic

Sets or updates the topic value in the payload of the CEvent object.

Request arguments

Name	Type	Description
topic	String	Name of a topic to set or update in the payload data.

Return parameter

None.

Request example

Example request to close an alert in a non-standalone Moolet:

```
moogdb.closeAlert(alert.value("alert_id"));
alert.setTopic("alerts.close");
alert.forward(this);
```

stringValue

Retrieves a value from inside the payload which matches the provided key as a string value.

Request arguments

Name	Type	Description
key	String	Key for a value stored in the payload which will be used to fetch the data.

Return parameter

Type	Description
String	Value from the payload that was stored alongside the key, or null if no value was found for the provided key, converted to string format.

type

Retrieves the type stored on the CEvent, this value indicates type of information in the payload and/or which topic the data came from.

Request arguments

None.

Return parameter

Type	Description
EBotEvent	Enum value specifying the type of data that the Event contains and/or which topic the data was received on from the bus.

value

Retrieves a value from inside the payload which matches the provided key. Objects such as custom_info are stored as JSON strings, not native objects. To return custom_info as a native JS object, use the getCustomInfo call instead.

Request arguments

Name	Type	Description
key	String	Key for a value stored in the payload which will be used to fetch the data.

Return parameter

Type	Description
String, Number or Boolean	Value from the payload that was stored alongside the key, or null if no value was found to for the provided key. Values are returned in their native stored format, that is, as a string, number, or Boolean. Native JS objects such as custom_info are stored in CEvent objects as JSON strings, and are returned as such by this method.

Events (MoogDb Only)

Compatibility with MoogDb and MoogDb.V2

Methods and auxiliary objects listed here are compatible with the MoogDb module, which was removed in v4.1.14.

Information here is provided for reference only.

For methods and auxiliary objects compatible with its replacement, see the MoogDb V2 module.

Description

The events Moobot module allows you to make a Moobot driven by the occurrence of events by defining the type of event that interests the Moobot.

The events module is available to load into any standard Moobot.

To use, at the top of a Moobot js file, define a new global variable **`events`** to load the events module:

```
var events = MooBot.loadModule('Events');
```

Method

- events.onEvent

The events module has only one method, **`onEvent`**. This method points the Moobot to a supplied JavaScript function, which is called when a specified event type occurs.

The parameters to the called function depend on the type of event that you are listening for. In a Moobot, this method is typically the last line in a script.

The type of event adaptor chosen is specific to the type of Moobot you are building.

events.onEvent

Takes the name of a valid JavaScript function in a Moobot and also event code (from the constants module **`eventType`**), and returns an event adaptor object.

Request Arguments

Name	Type	Description
`functionName`	String	The name of a valid JavaScript function in the Moobot that is called when the event arrives.

type	CEvent	**eventType** event code that specifies what type of event the Moobot is listening for. It is typically from the constants module.

Return Parameter

Name	Type	Description
CEventAdaptor	Object	An event adaptor object. Made active with the listen function in-line to listen for the event type.

Example

For the AlertBuilder MooBot:

```
events.onEvent("newEvent",constants.eventType("Event")).listen();
```

- Call the newEvent JavaScript function.
- Define the event type **event** (from the Constants module), which responds to events put on the Message Bus by a LAM.
- Call the listenfunction in-line to listen for the event type.

When the Moolet starts and loads this events Moobot, its JavaScript file executes, initialising the Moobot to respond in an event-driven way to events arriving.

newEvent Javascript function

The format of the function **newEvent** (which is called when you get an event), is as follows:

function newEvent

Request Arguments

Name	Type	Description
event	CEvent object	An object that encapsulates all the data for the event from the Message Bus.
response	CResponse object	An object to communicate back to the Moolet. The Moolet uses this response to broadcast any updates, or any changes to the data structures on the Message Bus.

CEventAdaptor auxiliary object

This object is a utility class used by the events module to allow for the programmatic activation of event listening. It has one method:

listen

Starts the event adaptor listening, which then calls the specified function when an event occurs.

Request Argument

None.

Return Parameter

Void - no value returned.

CEvent auxiliary object

This object encapsulates a generic Message Bus event object, and the contents of it are specific to the event type it represents. You can however access the key-value pairs contained in the object, and also set the values. Its methods include:

contains

Returns true if the event contains a value stored at the key name.

Request Argument

Name	Type	Description
name	String	The name of the key being queried.

Return Parameter

Type	Description
Boolean	True if the Event has a field called **name**, otherwise false.

set

Associates the specified **value** with the specified **name** in the event. Previous key mapping has the old **value** replaced.

Request Argument

Name	Type	Description
name	String	The key with which the specified value is to be associated.
value	Object	The value associated with the key.

Return Parameter

Type	Description
Boolean	Indicates if the operation was successful: true = success, false = fail.

value

Returns the object stored at the key **name**.

Request Argument

Name	Type	Description
name	String	The name of the key to return the object from.

Return Parameter

Type	Description
Object	A Javascript object containing what is at the key name.

CEvent auxiliary object

Note

The following methods only apply to the MoogDb module, which is being deprecated.

getCorrelationInfo

Returns the external service **correlation_info** (where this has been set) for a Situation.

Request Argument

Name	Type	Description
scope	Javascript object	The Moobot context, provided by using **this** as a parameter.

Return Parameter

Type	Description
Object	Javascript object containing the **correlation_info.**

getCustomInfo

Returns the custom information (if any) for an alert or Situation.

Request Argument

Name	Type	Description
scope	Javascript object	The Moobot context, provided by using this as a parameter.

Return Parameter

Type	Description
Object	Javascript object containing the custom information.

getJournalDetails

Returns the details (if any) of the journaled operation for a Situation.

Request Argument

Name	Type	Description
scope	Javascript object	The Moobot context, provided by using **this** as a parameter.

Return Parameter

Type	Description
Object	Javascript object containing the details of the journaled operation for a Situation.

getSummaryData

Returns the summary information from a statistics summary event.

Request Argument

Name	Type	Description
scope	Javascript object	The Moobot context, provided by using **this** as a parameter.

Return Parameter

Type	Description
Object	Javascript object containing the summary information.

setCustomInfo

Sets the custom information for an Alert or Situation

Request Arguments

Name	Type	Description
scope	Javascript object	The Moobot context, provided by using **this** as a parameter.
customInfoJS	Native object	A Javascript object containing the custom information.

Return Parameter

Void - no value returned.

CResponse auxiliary object

Note

The following methods only apply to the MoogDb module, which has been deprecated.

doNotPropagate

Indicates that no propagation is needed.

Request Argument

None.

Return Parameter

Void - no value returned.

message

Object to broadcast on.

Request Argument

Name	Type	Description
msg	CEvent	Object to broadcast on.

Return Parameter

Void - no value returned.

output

Freeform message to attach.

Request Argument

Name	Type	Description
txt	String	The message as a text string.

Return Parameter

Void - no value returned.

retcode

The **retcode** value must be >= 0 for a message to be sent.

Request Argument

Name	Type	Description
code	Number	Must be >= 0 for a message to be sent.

Return Parameter

Void - no value returned.

topic

Topic to broadcast message on.

Request Argument

Name	Type	Description
topic	String	The topic name.

Return Parameter

Void - no value returned.

Expose Active Moolets

You can expose which Moolets are running by adding functions to a Moobot. The functions are:

- Bot.isActive: Returns whether the specified Moolet is active or not.
- Bot.getActiveMoolets: Returns a list of all active Moolets in the system.

isActive

Returns whether the specified Moolet is active or not.

Request Argument

Name	Type	Required	Description
<mooletName>	String	Yes	Name of a Moolet.

Return Parameter

Type	Description
Boolean	'true' indicates the Moolet is active, 'false' indicates it is inactive.

Example

For example, you could use the function to return a logger warning if the ServiceNow Moolet is not running:

```
if(Bot.isActive('ServiceNow'))
        {
                var inform = mooletInforms.create('ServiceNow');
                inform.setSubject("ticket");
                inform.setDetails({sig_id: sigId}); inform.send();
        }
else
        {
                logger.warning("ServiceNow is not running - situation " + sigId
+ " was not sent");
        }
```

getActiveMoolets

Returns a list of all active Moolets in the Cisco Crosswork Situation Manager system.

Request Argument

None.

Return Parameter

Type	Description
List	A list of all active Moolets in the Cisco Crosswork Situation Manager system.

Example

You could use the function to return which Moolets are running if a specified Alert Workflow Engine Moolet is active:

```
var alert = moogdb.createAlert(event);
if(alert)
        {
                logger.info("New Alert Id: " + alert.value("alert_id"));
                if(Bot.isActive('AlertWorkflows'))
                {
                        logger.warning("Moolets running are: \n" +
Bot.getActiveMoolets());
                }
        }
```

An example log might return as follows:

```
WARN : [3:AlertBuilder][20190301 19:05:20.808 +0000] [AlertBuilder.js:128]
+|Moolets running are: [MaintenanceWindowManager, TeamsMgr, AlertBuilder,
SituationWorkflows, Housekeeper, Default Cookbook, Indexer, EnrichmentWorkflows,
AlertWorkflows, EventWorkflows, SituationMgr, SituationRootCause]|+
```

ExternalDb

The ExternalDb Moobot module allows Cisco Crosswork Situation Manager to access the following external relational databases, as well as any relational database that supports JDBC:

- MySQL
- Microsoft SQL Server
- IBM DB2
- Oracle
- PostgreSQL

Using ExternalDb, Cisco Crosswork Situation Manager can retrieve information from external databases for use in alerts and Situations. The ExternalDb method can also update external databases with information from Cisco Crosswork Situation Manager.

Load the module

The ExternalDb Moobot module is available to load into any standard Moobot. To load it, define a new global variable at the top of the Moobot Javascript file. For example:

```
var externalDb = MooBot.loadModule('ExternalDb');
```

Method reference

The ExternalDb module uses the following methods.

connect

Establishes a connection to an external database with defined connection properties.

Request arguments

The method takes the following arguments.

Name	Type	Required
properties	Object	Yes

Database connection properties

The properties object is a Javascript object that can contain the following keys. You can also define connection properties in the file **moog_external_db_details.conf**.

Key	Description
type	Database type. If you omit type you must specify the URL, jar files and JDBC class name. To use an external database other than those in the supported list, omit the **type** from the connection properties.
host	Database host name or IP address. Default is localhost.
database	Database name.
port	Port number. Default values: MySQL: 3306 SQL Server: 1433 DB2: 50000 Oracle: 1521 PostgreSQL: 5432
user	Username to connect to the database. If omitted you can specify it in the URL (for some databases) or the properties.
password	Password to connect to the database. If omitted you can specify it in the URL (for some databases) or the properties.
encrypted_password	Encrypted version of the password.
properties	A map of key-value pairs of properties to specify the connection properties. For example, **loginTimeout** for SQL Server or **useCompression** for MySQL.
jar_files	The JDBC driver jar file locations. Default values: SQL Server: sqljdbc4.jar DB2: db2jcc4.jar Oracle: ojdbc6.jar PostgreSql: postgresql-9.3-1102.jdbc41.jar

	The assumed location is **$MOOGSOFT_HOME/lib/cots/**. These files are not bundled in a standard Cisco Crosswork Situation Manager installation.
class_name	The name of the JDBC class. Default values: SQL Server: com.microsoft.sqlserver.jdbc.SQLServerDriver MySQL: com.mysql.jdbc.Driver DB2: com.ibm.db2.jcc.DB2Driver Oracle: oracle.jdbc.OracleDriver PostgreSql: org.postgresql.Driver
URL	JDBC-specific URL. If specified, it can override other properties.
pool_properties	A map of key-value pairs of properties of the connection pool that will be created. You can define the number of connections made available to the external database by including the **pool_size** key. **pool_size**: Number of connections in the pool. Must be 1 or more. Default is 10. Generally, this should match the number of threads configured to run the Moobot.

Response

The method returns the following parameter.

Type	Description
Object	A Java object containing the connection details. Returns **null** if the connection fails.

Examples

The method can accept a single parameter with connection properties, or two parameters: one with the generic connection properties and one specific to this connection.

The following command establishes a connection using the details in the **customers** object:

```
var customersConnection = externalDb.connect(dbTypes.customers);
```

The following command establishes a connection using the same details, but the **username** and **password** in the object are overridden by the ones provided in the command:

```
var customersConnection = externalDb.connect(dbTypes.customers, {user: 'myuser',
password: 'mypassword'});
```

execute

Performs an SQL update on the external database.

Request arguments

The method takes the following arguments.

Name	Type	Required
argument	String	Yes

Response

The method returns the following parameter.

Type	Description
Boolean	Indicates if the operation was successful: **true** = success, **false** = fail.

Examples

The following example uses the **execute** method to update the species column in a pets database.

```
employeesConnection.execute('update pets set species="dog" where species null');
```

query

Performs an SQL query on the external database.

Request arguments

The method takes the following arguments.

Name	Type	Required
argument	String	Yes

Response

The method returns the following parameter.

Type	Description
Object	A Java object containing the query results. Returns **null** if the connection fails.

Response methods

The response can contain the following methods.

Name	Description
rows	Returns the number of rows.
next	Returns the next row.
rewind	Goes back to the first row.
hasNext	Indicates whether the current row is the last one.
row(i)	Returns row **i** (zero based index).
first	Returns the first row.
type(name)	Returns the type of column called **name** (or **null** if no such column exists).
columnName(i)	Returns the name of column **i** (zero based index).
isNumber(name)	Returns **true** if the column name is a numeric column.
isString(name)	Returns **true** if the column name is a not numeric.

Row methods

You can use the following row methods in the query.

Name	Description

value(name)	Returns the value for column named name as a string.
columns	Returns the number of columns.
rewind	Goes back to the first column.
hasNext	Indicates whether the current column is the last one.
next	Returns the value in the next column as a string.
column(i)	Returns the value in column **i** as a string.
first	Returns the value in the first column as a string.
last	Returns the value in the last column as a string.

Example

The following example uses the **query** method to query the customers database.

```
var customers = customersConnection.query('Select * from customers');
while(customers.hasNext()==true)
{
    var customer = customers.next();
    var firstName = customer.value("first_name");
    var lastName = customer.value("last_name");
    logger.info(firstName + " " + lastName +" is a customer");
}
```

prepare

Performs SQL queries and updates on the external database. You can use the prepare method for complicated operations. For example, you can reuse the same SQL statement with different arguments, and you can use external data within a statement.

Request arguments

The method takes the following arguments.

Name	Type	Required
argument	String	Yes

Response

The method returns the following parameter.

Type	Description
Object	A Java object containing the query results. Returns **null** if the connection fails.

Response methods

The response object can contain the following methods.

Name	Description
set(i, value)	Sets parameter **i** (1 based index) to a value. Returns **false** in case of failure.
bind(value1, value2,	Sets parameter 1 to value 1, parameter 2 to value 2 and so on. Returns **false**

value3,...)	in case of failure.
bindCount	Returns the number of parameters needed to bind. Some vendors do not support this method in all cases, if not supported **-1** is returned.
execute(value1, value2, value3,...)	Sets parameter 1 to value 1, parameter 2 to value 2 and so on, and then executes the prepared statement. Returns false in case of a failure in one of the stages. If values are omitted, uses the previously **set** or **bind**.
query(value1, value2, value3,...)	Sets parameter 1 to value 1, parameter 2 to value 2 and so on, and then performs the query. Returns **null** in case of failure. Returns a result set if the operation was successful. If values are omitted, uses the previously **set** or **bind**.
close	Closes the prepared statement. Note: It is important to close the statement with this method when no longer needed.

Example

The following example uses the **prepare** method to set all pet species to "dog" if the breed is one of those specified:

```
var petsChange = employeesConnection.prepare('Update pets set species=? where
breed = ?');
petsChange.set(1, 'dog');
for (var breed in ['Labrador', 'Terrier', 'Beagle', 'Boxer', 'Poodle'])
{
    petsChange.set(2, breed);
    petsChange.execute();
}
petsChange.close();
```

Database drivers and declarations

When downloading JDBC drivers:

- Be sure to download the correct version of the driver for your database.
- Copy or move the downloaded drivers to **$MOOGSOFT_HOME/lib/cots/**.

Microsoft SQL Server

JDBC driver: http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=11774

Connection properties: http://technet.microsoft.com/en-us/library/ms378672(v=sql.110).aspx

Example declarations:

```
testdb:
{
    type: 'sqlServer',
    host: '172.16.87.248',
    port: '1433',
    database: 'my_db',
    user: 'myuser',
    password: 'mypassword'
}

testdb:
{
    jar_files: ["/usr/share/moogsoft/lib/cots/sqljdbc4.jar"],
```

```
    class_name: "com.microsoft.sqlserver.jdbc.SQLServerDriver",
    url: "jdbc:sqlserver://172.16.87.248:1433;databaseName=my_db",
    properties: { user: "myuser", password: "mypassword" }
}
```

MySQL

JDBC driver: Already included in Cisco Crosswork Situation Manager - no need to download.

Connection properties: http://dev.mysql.com/doc/connector-j/en/connector-j-reference-configuration-properties.html

Example declarations:

```
testdb:
{
    type: 'mySql',
    host: '172.16.87.247',
    port: '3306',
    database: 'my_db',
    user: 'myuser',
    password: 'mypassword'
}

testdb:
{
    jar_files: ["/usr/share/moogsoft/lib/cots/mysql-connector-java-5.1.37-
bin.jar"],
    class_name: "com.mysql.jdbc.Driver",
    url: "jdbc:mysql://172.16.87.247:3306/my_db",
    properties: { user: "myuser", password: "mypassword" }
}
```

IBM DB2

JDBC driver: http://www-01.ibm.com/support/docview.wss?uid=swg21363866

Connection properties: http://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.1.0/com.ibm.db2.udb.apdv.java.doc/doc/tjvjcccn.htm?cp=SSEPGG_9.1.0%2F8-1-4-2-1-0

Example declarations:

```
testdb:
{
    type: 'db2',
    host: '172.16.87.248',
    port: '50000',
    database: 'my_db',
    user: 'myuser',
    password: 'mypassword'
}

testdb:
{
    jar_files: ["/usr/share/moogsoft/lib/cots/db2jcc4.jar"],
    class_name: "com.ibm.db2.jcc.DB2Driver",
    url: "jdbc:db2://172.16.87.248:50000/my_db",
```

```
    properties: { user: "myuser", password: "mypassword" }
}
```

Oracle

JDBC driver: http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html

Connection properties: http://docs.oracle.com/cd/B28359_01/java.111/b31224/urls.htm

Example declarations:

```
testdb:
{
    type: 'oracle',
    host: '172.16.87.248',
    port: '1521',
    database: 'my_db',
    user: 'myuser',
    password: 'mypassword'
}

testdb:
{
    jar_files: ["/usr/share/moogsoft/lib/cots/ojdbc6.jar"],
    class_name: "oracle.jdbc.OracleDriver",
    url: "jdbc:oracle:thin:System/myuser@172.16.87.248:1521:my_db"
}
```

PostgreSql

JDBC driver: https://jdbc.postgresql.org/download.html

Connection properties: http://jdbc.postgresql.org/documentation/head/connect.html

Example declarations:

```
testdb:
{
    type: 'postgresql',
    host: '172.16.87.248',
    port: '5432',
    database: 'my_db',
    user: 'myuser',
    password: 'mypassword'
}

testdb:
{
    jar_files: ["/usr/share/moogsoft/lib/cots/postgresql-9.3-1102.jdbc41.jar"],
    class_name: "org.postgresql.Driver",
    url: "jdbc:postgresql://172.16.87.248:5432/my_db",
    properties: { user: "myuser", password: "mypassword" }
}
```

Graph Topology

The Graph Topology Moobot module allows you to create, manage and query multiple named topologies, and their nodes and links.

If your topology .csv file is larger than 40 MB Moogsoft recommends using the Topology Loader utility. See Load a Topologyfor information on the loader utility.

Load the module

The Graph Topology module is available to load into any standard Moobot. To load it, define a new global variable at the top of the Moobot Javascript file. For example:

```
var graphtopo = MooBot.loadModule('GraphTopo');
```

Method reference

The Graph Topology module uses the following methods.

createTopologies

Creates one or more topologies.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topologies	Array of JSON objects	Yes	One or more **topology** objects containing topology properties.

Topology properties

The **topology** objects can contain the following properties:

Name	Type	Required	Description
name	String	Yes	Name of the topology. Must be less than 256 characters.
active	Boolean	No	Flag to set the topology to active (**true**) or inactive (**false**). Default is **false**.
description	String	No	Description of the topology. Must be less than 1001 characters.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing the details of any topologies that could not be created.

Example

Example function using the **createTopologies** method to create two topologies named "host" and "location":

```
function createTopologies()
{
    logger.warning("Creating topologies");
    var topo1 =
    {
         name: "host",
         description: "Host-based topology",
         active: true
    };
```

```
        var topo2 =
        {
             name: "location",
             description: "Location-based topology",
             active: true
        };
        var topologiesNotCreated = graphtopo.createTopologies([topo1, topo2]);
        logger.warning("Returned object to string from createTopologies: " +
    JSON.stringify(topologiesNotCreated));
    }
```

updateTopologies

Updates one or more topologies.

You can also use this function to create a topology. If **name** does not exist the endpoint creates it.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topologies	Array of JSON objects	Yes	One or more **topology** objects containing topology properties.

Topology properties

The **topology** objects can contain the following properties:

Name	Type	Required	Description
name	String	Yes	Name of the topology. Must be less than 256 characters. You cannot update the topology name. To rename a topology use the **replaceTopology** method.
active	Boolean	No	Flag to set the topology to active (**true**) or inactive (**false**).
description	String	No	Description of the topology. Must be less than 1001 characters.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing the details of any topologies that could not be updated.

The request fails if any of the following are true:

1. The topology you're trying to update does not exist.
2. The topology you're trying to update is being used to filter a Recipe and you are trying to make it inactive.

Example

Example function using the **updateTopologies** method to update the descriptions of two topologies named "host" and "location":

```
function updateTopologies()
{
    logger.warning("Updating topologies");
```

```
        var topo1 =
        {
            name: "host",
            description: "Host-based network topology"
        };
        var topo2 =
        {
            name: "location",
            description: "Location-based network topology"
        };
        var topologiesNotUpdated = graphtopo.updateTopologies([topo1, topo2]);
        logger.warning("Returned object to string from updateTopologies: " +
    JSON.stringify(topologiesNotUpdated));
    }
```

deleteTopology

Deletes a single topology. Note the following:

- You cannot delete a topology if it's being used to filter a Recipe.
- Deleting a topology will impact your ability to restore associated Recipes in future. In order to restore a Recipe that filters on a named topology, the topology must still exist in Cisco Crosswork Situation Manager.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
name	String	Yes	Name of the topology.

Response

The method returns the following response.

Type	Description
Boolean	**true**: Topology was deleted. **false**: Topology was not deleted.

The request fails if **name** is being used to filter a Recipe, or does not exist.

Example

Example function using the **deleteTopology** method to delete a topology named "host":

```
    function deleteTopo(host)
    {
        var deleteTopoResult = graphtopo.deleteTopology(host);
        logger.warning("Returned object to string from deleteTopo: " +
    JSON.stringify(deleteTopoResult));
    }
```

getTopology

Retrieves a single topology.

You can also use this function to determine whether a topology exists. If **`getTopology`** returns null, the specified topology does not exist.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
`name`	String	Yes	Name of the topology.

Response

The method returns the following response.

Type	Description
JSON object	Object containing the topology details.

Example

Example request using the **`getTopology`** method to retrieve a topology named `"host"`:

```
var topology = graphtopo.getTopology(host);
```

The response object contains details of the `"host"` topology:

```
{
    name: "host",
    description: "Host-based topology",
    active: true
}
```

getActiveTopologies

Retrieves all active topologies.

Request arguments

The method takes no request arguments.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing the details of active topologies.

Example

Example request using the **`getActiveTopologies`** method to retrieve all active topologies:

```
var active = graphtopo.getActiveTopologies();
```

The response object contains details of all active topologies:

```
[
    {
        name: "host",
        description: "Host-based topology",
        active: true
    },
    {
        name: "location",
```

```
            description: "Location-based topology",
            active: true
        }
    ]
```

getInactiveTopologies

Retrieves all inactive topologies.

Request arguments

The method takes no request arguments.

Response

The method returns the following response.

Type	Description
JSON object	Object containing the details of all inactive topologies.

Example

Example request using the **`getInactiveTopologies`** method to retrieve all inactive topologies:

```
var inactive = graphtopo.getInactiveTopologies();
```

The response object contains details of all inactive topologies:

```
[
    {
        name: "virtual",
        description: "Virtual topology",
        active: false
    },
    {
        name: "physical",
        description: "Physical topology",
        active: false
    }
]
```

getTopologiesForSituation

Retrieves all topologies linked to a specified Situation.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
`sitn_id`	Number	Yes	ID of the Situation for which to retrieve topologies.

Response

The method returns the following response.

Type	Description
Array of	One or more objects containing the details of topologies linked to the Situation, including

JSON objects	topology name and a flag indicating whether this topology caused the Situation to be created.

Example

Example request using the **`getTopologiesForSituation`** method to retrieve all topologies linked to Situation 12:

```
var situationtopos = graphtopo.getTopologiesForSituation(12);
```

The response object contains details of all linked topologies and their causal flags:

```
[
    {
        "name" : "host",
        "causal" : true
    },
    {
        "name" : "location",
        "causal" : false
    }
]
```

cloneTopology

Clones an existing topology and sets the clone to inactive.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
`topoName`	String	Yes	Name of the topology to clone.
`cloneTopoName`	String	Yes	Name for the cloned topology. Must be less than 256 characters.

Response

The method returns the following response.

Type	Description
JSON object	Object containing details of the cloned topology.

Example

Example function using the **`cloneTopology`** method to clone the `"host"` topology and name the clone `"host_new"`:

```
function cloneTopology(host, host_new)
{
    logger.warning("Calling function cloneTopology for topo: [" + host + "] -
new name will be: [" + host_new + "]");
    var clonedTopology = graphtopo.cloneTopology(host, host_new);
    logger.warning("Returned object to string from cloneTopology: " +
JSON.stringify(clonedTopology));
}
```

The response contains details of the `"host_new"` topology:

```
{
    "name" : "host_new",
    "description" : "host-based topology",
```

```
    "active": false
}
```

replaceTopology

Replaces an existing topology with another topology. This process deletes the original topology. You can use the clone and replace Graph Topology methods to update a copy of an existing topology and then replace a topology with the updated version.

You can also use this method to rename a topology.

When a topology is replaced:

- The original topology and its nodes and links are deleted.
- Alerts that reference the original topology are updated to reference the replacement topology.
- If the replacement topology is active, its processing state in the database is set to outdated. This triggers the graph analyser process to run as part of the Housekeeper Moolet to calculate Vertex Entropy for the topology nodes. See Topologies.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Replace: The name of the existing topology to replace. Rename: The new topology name.
topology	JSON object	Yes	Object containing properties of the replacing topology, or the topology to rename.

Topology properties

The **topology** object contains the following properties:

Name	Type	Required	Description
name	String	Yes	Replace: Name of the replacing topology. Rename: The topology to rename.
active	Boolean	No	Sets the replaced or renamed topology to active (**true**) or inactive (**false**). Replaced topologies take the active status of the replacing topology by default.

Response

The method returns the following response.

Type	Description
JSON object	Object containing details of the newly replaced or renamed topology.

The request fails if any of the following are true:

- **name** is being used to filter a Recipe, or does not exist.

- **topoName** is being used to filter a Recipe and you are trying to make it inactive.

Example

Example function using the **replaceTopology** method to replace the "host" topology with the "host_new" topology and set its status to active:

```
function replaceTopology(host, host_new)
{
    logger.warning("Calling function replaceTopology - topology to replace: [" +
host + "], replacing topology: [" + host_new + "]");
    var replacingTopology = { name: host_new, active: true };
    var replacedTopology = graphtopo.replaceTopology(host, replacing_topology);
    logger.warning("Returned object to string from replaceTopology: " +
JSON.stringify(replacedTopology));
}
```

In this example, if there is no topology named "host" the "host_new" topology is renamed "host".

createNodes

Creates nodes in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology in which to create the nodes.
nodes	Array of JSON objects	Yes	One or more **node** objects containing node properties.

Node properties

The **node** objects can contain the following properties:

Name	Type	Required	Description
name	String	Yes	Name of the node. Must be less than 256 characters.
description	String	No	Description of the node. Must be less than 1001 characters.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing the details of any nodes that could not be created.

Example

Example function using the **createNodes** method to create two nodes in the "host" topology named "node1" and "node2":

```
function createNodes()
{
    logger.warning("Creating nodes");
    var node1 =
    {
        name: "node1",
        description: "First node"
    };
```

```
        var node2 =
        {
            name: "node2",
            description: "Second node"
        };
        var nodesNotCreated = graphtopo.createNodes(host, [node1, node2]);
        logger.warning("Returned object to string from createNodes: " +
    JSON.stringify(nodesNotCreated));
    }
```

updateNodes

Updates specified nodes in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology in which to create the nodes.
nodes	Array of JSON objects	Yes	One or more **node** objects containing node properties.

Node properties

The **nodes** objects can contain the following properties:

Name	Type	Required	Description
name	String	Yes	Name of the node. You cannot update the node name. To rename a node delete it and recreate it with a new name.
description	String	No	Description of the node. Must be less than 1001 characters.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing the details of any nodes that could not be updated.

Example

Example function using the **updateNodes** method to update two nodes in the "host" topology named "node1" and "node2":

```
function updateNodes()
{
    logger.warning("Updating nodes");
    var node1 =
    {
        name: "node1",
        description: "Primary node"
    };
    var node2 =
    {
        name: "node2",
        description: "Secondary node"
```

```
    };
    var nodesNotUpdated = graphtopo.updateNodes(host, [node1, node2]);
    logger.warning("Returned object to string from updateNodes: " +
JSON.stringify(nodesNotUpdated));
}
```

deleteNodes

Deletes specified nodes in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology that contains the nodes to delete.
nodes	Array of Strings	Yes	Names of one or more nodes to delete.

Response

The method returns the following response.

Type	Description
Array of Strings	Names of one or more nodes that could not be deleted.

Example

Example function using the **deleteNodes** method to delete two nodes in the "host" topology named "node1" and "node2":

```
function deleteNodes()
{
    logger.warning("Deleting nodes");
    var nodesNotDeleted = graphtopo.deleteNodes(host, ["node1", "node2"]);
    logger.warning("Returned object to string from deleteNodes: " +
JSON.stringify(nodesNotDeleted));
}
```

getNode

Retrieves details of a single node in a specified topology.

You can also use this function to determine whether a node exists. If **getNode** returns null, the specified node does not exist.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology containing the node.
nodeName	String	Yes	Name of the node.

Response

The method returns the following response.

Type	Description
JSON object	Object containing the node details.

Example

Example using the **getNode** method to return details of "node1" in the "host" topology.

```
var node1 = graphtopo.getNode("host", "node1");
```

The response object contains the node details:

```
{
    name: "node1",
    description: "First node"
}
```

getAllNodes

Retrieves details of all nodes in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology containing the nodes.

Response

The method returns the following response.

Type	Description
JSON object	Object containing details of the nodes.

Example

Example using the **getAllNodes** method to return details of all nodes in the "host" topology:

```
var allNodes = graphtopo.getAllNodes("host");
```

The response object contains details of the nodes in the "host" topology:

```
[
    {
        name: "node1",
        description: First node
    },
    {
        name: "node2",
        description: Second node
    }
]
```

isConnected

Checks whether a specified node is part of a topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology that contains the node.
nodeName	String	Yes	Name of the node.

Response

The method returns the following response.

Type	Description
Boolean	**true**: Node in topology. **false**: Node not in topology.

Example

Example using the **isConnected** method to check whether the "node1" node is in the "host" topology.

```
var isNodeInTopo = graphtopo.isConnected("host", "node1");
```

numberOfConnections

Counts the number of links from a specified node in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology that contains the node.
nodeName	String	Yes	Name of the node for which to count the links.

Response

The method returns the following response.

Type	Description
Number	The number of connections from the node. Returns **0** if the node does not exist or has no connections, or the topology does not exist.

Example

Example using the **numberOfConnections** method to return the number of connections from node "node1" in the "host" topology:

```
var return = topo.numberOfConnections("host", "node1");
logger.warning("numberOfConnections -> " + return);
```

This example logs the following message:

```
WARN : ... [CLogModule.java]:99 +|numberOfConnections -> 3|+
```

distance(topoName, sourceNode, sinkNode)

Calculates the number of hops between two specified nodes in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology that contains the nodes.
sourceNode	String	Yes	Name of the source node. Must be less than 256 characters. Note that topology links in Cisco Crosswork Situation Manager are bidirectional. If the node does not exist, the endpoint will create it.
sinkNode	String	Yes	Name of the sink node. Must be less than 256 characters. Note that topology links in Cisco Crosswork Situation Manager are bidirectional. If the node does not exist, the endpoint will create it.

Response

The method returns the following response.

Type	Description
Double	The number of hops between the nodes. Returns **-1** if the nodes are not connected.

Example

Example function using the **distance** method to calculate the number of hops between nodes "node1" and "node2" in the "host" topology:

```
function distance(host, node1, node2)
{
    logger.warning("Calling function distance for topo: [" + host + "] and
nodes: [" + node1 + "], [" + node2 + "]");
    var distance = graphtopo.distance(host, nost1, node2);
    logger.warning("Returned object to string from distance: " +
JSON.stringify(distance));
}
```

distance(topoName, sourceNode, sinkNode, radius)

Calculates the number of hops between two specified nodes in a specified topology, up to a maximum number of hops.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology that contains the nodes.
sourceNode	String	Yes	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Yes	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
radius	Double	Yes	Maximum number of hops to count.

Response

The method returns the following response.

Type	Description
Double	The number of hops between the nodes. Returns **-1** if the nodes are not connected or the number of hops is larger than the radius.

Example

Example function using the **distance** method to calculate the distance between nodes "node1" and "node2" in the "host" topology, where the number of hops is 5 or less:

```
function distance(host, node1, node2, 5.0)
{
    logger.warning("Calling function distance (with max hops) for topo: [" +
host + "] and nodes: [" + node1 + "], [" + node2 + "] and radius: [5.0]");
    var distanceWithMaxHops = graphtopo.distance(host, node1, node2, 5.0);
    logger.warning("Returned object to string from distance (with max hops): " +
JSON.stringify(distance));
}
```

createLinks

Creates links between specified nodes in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology in which to create the links.
links	Array of JSON objects	Yes	One or more **link** objects containing link details.

Link properties

The **link** objects can contain the following properties:

Name	Type	Required	Description
sourceNode	String	Yes	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Yes	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
description	String	No	Description of the link. Must be less than 1001 characters.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing the details of any links that could not be created.

Example

Example function using the **createLinks** method to create links "node1" > "node2" and "node2" > "node3" in the "host" topology:

```
function createLinks()
{
    logger.warning("Creating links");
    var link1 =
    {
```

```
            sourceNode: "node1",
            sinkNode: "node2",
            description: "link1"
        };
        var link2 =
         {
            sourceNode: "node2",
            sinkNode: "node3",
            description: "link2"
        };
        var linksNotCreated = graphtopo.createLinks(host, [link1, link2]);
        logger.warning("Returned object to string from createLinks: " +
    JSON.stringify(linksNotCreated));
    }
```

getLink

Retrieves details of a link between specified nodes in a specified topology.

You can also use this function to determine whether a link exists. If **`getLink`** returns null, the specified link does not exist.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
`topoName`	String	Yes	Name of the topology containing the link.
`sourceNode`	String	Yes	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
`sinkNode`	String	Yes	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.

Response

The method returns the following response.

Type	Description
JSON object	Object containing the link details.

Example

Example using the **`getLink`** method to return details of the link between `"node1"` and `"node2"` in the `"host"` topology.

```
var linkdetails = graphtopo.getLink("host", "node1", "node2");
```

The response object contains the link details:

```
{
    description: "link1",
    sourceNode: "node1",
    sinkNode: "node2"
}
```

getAllLinksForTopology

Retrieves all links for a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology containing the links.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing details of the links.

Example

Example using the **getAllLinksForTopology** method to retrieve details of all links in the "host" topology:

```
var linkDetails = graphtopo.getAllLinksForTopology("host");
```

The response object contains details of the links:

```
[
    {
        description: "link1",
        sourceNode: "node1",
        sinkNode: "node2"
    },
    {
        description: "link2",
        sourceNode: "node2",
        sinkNode: "node3"
    }
]
```

getAllLinksForNode

Retrieves all links for specified node in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
topoName	String	Yes	Name of the topology containing the links.
nodeName	String	Yes	Name of the node.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing details of the links.

Example

Example using the **`getAllLinksForNode`** method to retrieve link details for the `"node2"` node in the `"host"` topology:

```
var linksForNode = graphtopo.getAllLinksForNode("host", "node2");
```

The response object contains details of the links:

```
[
    {
        description: "link1",
        sourceNode: "node2",
        sinkNode: "node1"
    },
    {
        description: "link2",
        sourceNode: "node2",
        sinkNode: "node3"
    }
]
```

deleteAllLinksForNode

Deletes all links for a specified node in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
`topoName`	String	Yes	Name of the topology containing the links.
`nodeName`	String	Yes	Name of the node.

Response

The method returns the following response.

Type	Description
Boolean	**`true`**: Links were deleted. **`false`**: Links were not deleted.

Example

Example using the **`deleteAllLinksForNode`** method to delete all links for the `"node2"` node in the `"host"` topology:

```
var deleteLinks = graphtopo.deleteAllLinksForNode("host", "node2");
```

deleteLinks

Deletes one or more links in a specified topology.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description

topoName	String	Yes	Name of the topology that contains the links to delete.
links	Array of JSON objects	Yes	One or more **link** objects containing link details.

Link properties

The **link** objects can contain the following properties:

Name	Type	Required	Description
sourceNode	String	Yes	Name of the source node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.
sinkNode	String	Yes	Name of the sink node. Note that topology links in Cisco Crosswork Situation Manager are bidirectional.

Response

The method returns the following response.

Type	Description
Array of JSON objects	One or more objects containing the details of any links that could not be deleted.

Example

Example function using the **deleteLinks** method to delete two links in the "host" topology between "node1" ands "node2", and "node2" and "node3":

```
function deleteLinks()
{
    logger.warning("Deleting links");
    var link1 =
    {
        sourceNode: "node1",
        sinkNode: "node2"
    };
    var link2 =
    {
        sourceNode: "node2",
        sinkNode: "node3"
    };
    var linksNotDeleted = graphtopo.deleteLinks("host", [link1, link2]);
    logger.warning("Returned object to string from deleteLinks: " +
JSON.stringify(linksNotDeleted));
}
```

Deprecated methods

The following Topology methods are no longer supported.

loadTopology

Use getTopology.

isConnected(node)

Use getNode.

connected(node1, node2)

Use isConnected.

numberOfConnections(node)

Use numberOfConnections.

addEdge(node1, node2)

Use createLinks.

addEdge(node1, node2, weight)

Use createLinks.

distance(node1, node2)

Use distance(topoName, sourceNode, sinkNode).

distance(node1, node2, radius)

Use distance(topoNode, sourceNode, sinkNode, radius).

radialDelta(node1, node2)

Use distance(topoName, sourceNode, sinkNode).

Kafka

The Kafka module allows you to broadcast information on a Kafka bus. For example, you can use it to push alert or Situation data to a data lake via Kafka.

Load the module

The Kafka Moobot module is available to load into any standard Moobot. To load it, define a new global variable at the top of the Moobot Javascript file. For example:

```
var kafka = MooBot.loadModule('Kafka');
```

Method reference

The Kafka module uses the following methods.

connect

Establishes a connection to one or more Kafka brokers with defined connection properties.

Request arguments

The method takes the following arguments.

Name	Type	Required
properties	Object	Yes

Connection properties

The **properties** object can include the following keys:

Name	Type	Required	Description
servers	List	Yes	Specifies the Kafka broker or

			brokers to connect to.
connection_id	String	Yes	Name of the connection.
use_ssl	Boolean	Yes	Whether to connect using SSL.
ssl_truststore_location	String	No	Path to the SSL truststore file.
ssl_truststore_password	String	No	Password for the SSL truststore file.
ssl_truststore_encrypted_password	String	No	Encrypted password for the SSL truststore file.
ssl_key_password	String	No	Password for the SSL certificate.
ssl_key_encrypted_password	String	No	Encrypted password for the SSL certificate.
ssl_endpoint_identification_algorithm	String	No	Endpoint identification algorithm to validate the server hostname.
compression_codec	String	No	Compression codec to use. One of: **none** **gzip** **lz4** **snappy**
use_sasl	Boolean	No	Whether to connect using SASL.
sasl_mechanism	String	No	SASL mechanism to use. If you set a SASL mechanism you must set **use_sasl** to true. One of: **PLAIN** **SCRAM-SHA-256** **SCRAM-SHA-512** **OAUTHBEARER** Choose from **PLAIN**, **SCRAM-SHA-256**, **SCRAM-SHA-512** and **OAUTHBEARER**.
security_protocol	String	No	SASL protocol to use. If you set a SASL protocol you must set **use_sasl** to true. One of: **SASL_SSL** **SASL_PLAINTEXT**
sasl_jaas_config	String	No	Credentials for JAAS authentication.

additional_properties	Various	No	Kafka consumer properties. Any properties you define here take priority over any configurations. See the Apache Kafka documentation for descriptions of these properties.
sasl_kerberos_service_name	String	No	Kerberos service name.
kerberos_conf_file_path	String	No	Path to the Kerberos configuration file.
kerberos_debug_log	Boolean	No	Whether to enable Kerberos authentication debug logs.

Response

The method returns the following parameter.

Type	Description
Object	A Java object containing connection details, depending on the requested connection properties. Returns **null** if no connection can be made.

Example

Example use of the **connect** method without SSL:

```
var conn = kafka.connect(
{
    servers: ["my.kafka.broker:9092", "my.other.kafka.broker:9092"],
    connection_id: "KafkaConnection",
    use_ssl: false
});
```

Example use of the **connect** method using SSL:

```
var conn = kafka.connect(
{
    servers: ["my.kafka.broker:9092", "my.other.kafka.broker:9092"],
    connection_id: "KafkaConnection",
    use_ssl: true,
    ssl_truststore_location: "/path/to/truststore",
    ssl_truststore_password: "test1234",
    ssl_keystore_location: "/path/to/keystore",
    ssl_keystore_password: "test1234",
    ssl_key_password: "test1234",
    ssl_endpoint_identification_algorithm: "",
    use_sasl: false
});
```

Example use of the **connect** method using SASL:

```
var conn = kafka.connect(
{
    servers: ["my.kafka.broker:9092", "my.other.kafka.broker:9092"],
    connection_id: "KafkaConnection",
    use_ssl: false,
```

```
        use_sasl: true,
        sasl_mechanism: "PLAIN",
        security_protocol: "SASL_PLAINTEXT",
        sasl_jaas_config: "org.apache.kafka.common.security.plain.PlainLoginModule
    required username=myuser password=mypassword;",
        additional_properties:{
           "sasl.login.refresh.window.factor": "0.8",
           "sasl.login.refresh.window.jitter": "0.05",
           "sasl.login.refresh.min.period.seconds": "60",
           "sasl.login.refresh.min.buffer.seconds": "300"
         },
        sasl_kerberos_service_name: "kafka",
        kerberos_conf_file_path : "/etc/krb5.conf",
        kerberos_debug_log : true
    });
```

send

Sends a message to the Kafka broker.

Request arguments

The method takes the following arguments.

Name	Type	Required
topic	String	Yes
key	String	No
message	JSON	Yes

Response

The method returns the following parameter:

Type	Description
Boolean	Indicates whether the send operation was successful.

Example

Example use of the **send** method:

```
    if (connection)
    {
        connection.send("myTopic", {test_field: "value"});
        connection.send("myTopic", "myKey", {test_field: "value"});
    }
```

close

Closes the connection to the Kafka broker.

Request arguments

The method takes no arguments.

Response

None.

Examples

Example function using the Kafka module:

```
function sendToKafka(alert)
{
    var connection = kafka.connect({
        servers: ["my.kafka.broker:9092", "my.other.kafka.broker:9092"],
         connection_id: "KafkaConnection",
         use_ssl: false
    });

    if (connection){
        boolean success = connection.send("myTopic", {test_field: "value"});
         if (!success){
         logger.warning("Failed to send message to Kafka");
         }
    }
}
```

Logger

Warning

The Logger module was deprecated with the release of Cisco Crosswork Situation Manager v7.1. See Configure Logging for new logging details.Configure Logging

The Logger module sets the log level in Moogfarmd, allowing log messages to be written to the common Moogfarmd log file. See Configure Logging for more information on configuring logging.Configure Logging

When you create a Moobot you can use the Logger module for debugging. Writing a log message to a file is an IO operation, and comes with execution cost. When developing a Moobot it can be helpful to include a number of logging statements. Once the Moobot is operational, however, it is best to keep logging to a minimum.

Load the module

The Logger module is available to load into any standard Moobot. To load it, define a new global variable at the top of the Moobot Javascript file. For example:

```
var logger = MooBot.loadModule('Logger');
```

Method reference

The Logger module uses the following methods. Note that **`printf`** Logger functions have been deprecated in favour of the 'single string argument' version. For more information see the Wikipedia entry for Printf format string.

debug

Sends a debug log message (the lowest severity level). You can use the debug method to log detailed troubleshooting information in non-production environments.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description

logmessage	String	Yes	A log message formed of a single string of JavaScript variables or objects. Use concatenation to form multiple arguments.

Response

None.

Example

Example use of the **debug** method to log a message:

```
logger.debug("A debug message");
```

This example logs the following message:

```
DEBUG:... ...A debug message
```

info

Sends an information log message (the intermediate severity level). You can use it to log changing a property, for example.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
logmessage	String	Yes	A log message formed of a single string of JavaScript variables or objects. Use concatenation to form multiple arguments.

Response

None.

Example

Example use of the **info** method to log a message:

```
{
   var dispText= "Reset";
   var dispNum= 2;
   var aReal= 3.141593;

   logger.info("Counter: "+ dispText);
   logger.info("Severity low. Level: "+ dispNum + ". ...Pi = "+ aReal);
}
```

This example logs the following messages:

```
INFO :... ...Counter: Reset
INFO :... ...Severity low. Level: 2. ...Pi = 3.141593
```

warning

Sends a warning message (high severity level). You can use it to log behavior that impacts normal operation of the system.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description

logmessage	String	Yes	A log message formed of a single string of JavaScript variables or objects. Use concatenation to form multiple arguments.

Response

None.

Example

Example use of the **warning** method to log a message:

```
{
   var aString= "CPU@ >90%";
   var intHigh= 4;

   logger.warning("Warning: "+ aString);
   logger.warning("Severity high. Level: "+ intHigh);
}
```

This example logs the following messages:

```
WARN :... ...Warning: CPU@ >90%
WARN :... ...Severity high. Level: 4
```

fatal

Sends a fatal log message (the highest severity setting). You can use it to log extreme circumstances, such as an unrecoverable failure that caused Moogfarmd to exit.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
logmessage	String	Yes	A log message formed of a single string of JavaScript variables or objects. Use concatenation to form multiple arguments.

Response

None.

Example

Example use of the **fatal** method to log a message:

```
{
   var intHighest= 5;

   logger.fatal("Severity exceeds "+ intHighest + "! Restart required");
}
```

This example logs the following message:

```
FATAL:... ...Severity exceeds 5! Restart required
```

Mailer

The Mailer module allows you to send an email message in response to events occurring in Cisco Crosswork Situation Manager. For example, you can use it in the Notifier.js Moobot to send email to users when they are invited to a Situation Room.

Load the module

The Mailer module is available to load into any standard Moobot. To load it, define a new global variable at the top of the Moobot Javascript file. For example:

```
var mailer = MooBot.loadModule('Mailer');
```

Method reference

The Mailer module uses the following methods.

intTransport

Defines the mail server information needed to send the email in the send function.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
`mailerObj`	Object	Yes	A JSON object specifying connection properties.

Use the following port guidelines:

1. If using port 587, set **`start_tls`** to true and **`use_tls`** to false.
2. If using port 465, set **`start_tls`** to true and **`use_tls`** to true.
3. If using port 25, set **`start_tls`** to false and **`use_tls`** to false (or comment out both properties).

If you omit the **`password`** field, an unauthenticated connection is created between Mailer and the server.

Response

None.

Example

Example use of the **`intTransport`** method to specify a mailserver, port and other connection details:

```
mailer.initTransport(
{
    server          : "smtp.mailserver.com",
    port            : 2525,
    account         : "user@mailserver.com",
    password        : "m00gsoft",
    isEncrypted     : false,
    start_tls       : false,
    use_tls         : false
});
```

send

Sends the email message. You must define a callback function in the same Moobot that is referenced in the mailMsg executed after a successful transmission.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
mailerObj	Object	Yes	A JSON object containing the fields required to populate the email message.

Response

None.

Example

Example use of the **send** method to send an email message:

```
var mailMsg = {
    to      : "destination@mail.com",
    subject : "Situation Room Notification",
    message : "email body",
    invite  : invite, // do not change
    bot     : MooBot.self, // do not change
    callback: "sendSuccess", // the name of the function to run in this Moobot
    args    : [ invite_id, "Sent successfully", vector ] // do not change
    };
mailer.send(mailMsg);
```

Moolet Informs

The Moolet Informs module enables the exchange of messages between Moolets. You can configure a Moolet to update other Moolets. For example, after you label some alerts you can configure the module to inform the ticketing Moolet to update the severity of a ticket, based on the new label.

To use Moolet Informs, you need:

1. A Moolet and associated Moobot to send inform messages from.
2. One or more Moolets to receive the inform messages. These are your "targets".

Load the module

The Moolet Informs module is available to load into any standard Moobot. To load it, define a new global variable at the top at the top of the Javascript file of the Moobot associated with the Moolet you want to send messages from. For example:

```
var mooletInforms = MooBot.loadModule('MooletInforms');
```

Configure the Moobot as outlined below.

Configure the module

To configure the Moolet Informs module:

1. Create the Moolet Inform using the **create** method as follows, passing the target Moolets that receive messages from this source:

```
var inform = mooletInforms.create("AlertRulesEngine", "Cookbook");
```

Or, you can choose not to specify the target Moolets in this step:

```
var inform = mooletInforms.create();
```

2. Add values to the inform using one or more of the following:
    - inform.setSubject: Subject of the inform message. You can use this to enable a different workflow within the target Moobot.
    - inform.setPayload: Any CEvent object. See Events for more information.
    - inform.setDetails: Details of any other data you want to send as a JSON object.
3. If you did not specify the target Moolets previously, specify them now:
    - inform.setTarget: List of target Moolets to receive messages.
4. You can configure message sending in two ways. If you have already set your targets:
    - **inform.send();**

If you have not set your targets, include them in the method call:

- **inform.send("AlertRulesEngine", "Cookbook");**

5. Edit the configuration file for each target Moolet and add an event handler to listen for the Inform messages:

```
events.onEvent("informReceive",
constants.eventType("mooletInforms.ExampleMoolet")).listen();
```

6. The listening target Moolet can access the data in two ways. You can use the **getSubject**, **getPayload** and **getDetails** methods:

```
function informReceive(inform)
{
    var subject = inform.getSubject();
    var payload = inform.getPayload();
    var details = inform.getDetails();
    logger.warning("Received Moolet Inform. Subject [" + subject + "] Payload ["
+ payload + "] Details [" + details + "]");
}
```

Alternatively, you can use the **value** method:

```
function informReceive(inform)
{
    var subject = inform.value("subject");
    var payload = inform.value("payload");
    var details = inform.value("details");
    logger.warning("Received Moolet Inform. Subject [" + subject + "] Payload ["
+ payload + "] Details [" + details + "]");
}
```

7. You can configure the Moolet to call a specific method for different subjects in the inform messages. For example you can configure a Remedy Moolet to listen for a specific subject in the inform message and route the event to a function:

```
events.onEvent("createNewTicket",
constants.eventType("mooletInforms.RemedyMoolet.ticketCreate")).listen();
```

After you have completed your configuration, inform messages are sent to your target Moolets which will call any methods you have added.

Method reference

The Moolet Informs module uses the following methods.

create

Creates a Moolet inform message. You can choose to select one or more Moolet targets to receive the messages, or you can leave the method empty.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
targets	String	No	A single Moolet or comma separated list of Moolets to target.

Response

The method returns the following parameter.

Type	Description
Object	A Moolet inform Java object.

Example

The following example uses the **create** method to set MaintenanceWindowManager and AlertRulesEngine as targets:

```
var inform = mooletInforms.create("MaintenanceWindowManager",
"AlertRulesEngine");
```

setSubject

Sets the subject for Moolets to listen for on the Message Bus.

Request arguments

The method takes the following arguments:

Name	Type	Required	Description
Subject	String	Yes	Subject the Moolets listen for on the Message Bus.

Response

None.

Example

The following example uses the **setSubject** method to set the subject "subTopic":

```
inform.setSubject("subTopic");
```

setPayload

Sets the payload to a CEvent object. See Events for more information.

Request arguments

The method takes the following argument:

Name	Type	Required	Description
Payload	CEvent	Yes	A CEvent object that has been passed into the Moobot from the pipeline, or has been retrieved from MoogDb.

Response

None.

Example

The following example uses the **`setPayload`** method to set the payload to the `"event"` CEvent:

```
inform.setPayload(event);
```

setDetails

Sets any other details to send in the Moolet Inform message.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
`setDetails`	NativeObject	Yes	A JSON object containing details to send in the message.

Response

None.

Example

The following example uses the **`setDetails`** method to send signature, description and source details in the message.

```
inform.setDetails({"signature":"Loss of Signal","description":"Loss of
Signal","source":"S-DF_P2_1"});
```

setTarget

Sets the target Moolets to receive the Moolet Inform messages. Use this method if you did not set the target Moolets with the **`create`** method.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
`targets`	String	Yes	A single Moolet or comma separated list of Moolets to target.

Response

None.

Example

The following example uses the **`setTarget`** method to set AlertRulesEngine and Cookbook as targets:

```
inform.setTarget("AlertRulesEngine", "Cookbook");
```

send

Sends the Moolet Informs messages to the target Moolets.

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
targets	String	No	A single Moolet or comma separated list of Moolets to target.

Response

None.

Example

The following example uses the **send** method to send messages to the Cookbook:

```
inform.send("Cookbook");
```

Example

The following example uses the Moolet Inform module to send a signature, description and source to the Cookbook Moolet:

```
var mooletInforms = MooBot.loadModule('MooletInforms');
var inform = mooletInforms.create();
inform.setSubject("subTopic");
inform.setPayload(event);
inform.setDetails({"signature":"Loss of Signal","description":"Loss of
Signal","source":"S-DF_P2_1"});
inform.send("Cookbook");
```

Example configuration in the target Moolet:

```
events.onEvent("handleEvent",
constants.eventType("mooletInforms.EmptyMoolet.event_subject")).listen();
events.onEvent("handleAlert",
constants.eventType("mooletInforms.EmptyMoolet.alert_subject")).listen();
events.onEvent("handleSig",
constants.eventType("mooletInforms.EmptyMoolet.sig_subject")).listen();
```

Moolet Information API (Bot API)

You can use the Moolet Information API or Bot API methods in a Moobot to obtain contextual information about the associated Moolet. These methods are useful in automation and other workflows where you want to verify the Moolet context before performing an action such as sending data.

Method reference

The Moolet Information API uses the following methods.

getType

Retrieves the Moolet type.

Request arguments

The method takes no arguments.

Response

The method returns one of the following parameters.

Type	Description
Enumerated type	One of the following: ALERT_BUILDER ALERT_RULE_ENGINE COOKBOOK EMPTY_MOOLET NOTIFIER SCHEDULE SITUATION_MANAGER TEAMS_MANAGER WORKFLOW_ENGINE

If the result is WORKFLOW_ENGINE you can use the **`WorkflowEngine.getMessageType`** method to retrieve the Workflow Engine type.

Example

Example use of the **`getType`** method:

```
var MooletType = Bot.getType();
logger.warning(' Moolet type is ...' +MooletType);
```

getMooletName

Retrieves the Moolet name.

Request arguments

The method takes no arguments.

Response

The method returns the following parameter.

Type	Description
String	Name of the associated Moolet.

Example

Example use of the getMooletName method:

```
if((Bot.getType() === Bot.EMPTY_MOOLET ))
   {
      logger.warning(Bot.getMooletName() + ' is an empty moolet')
   };
```

WorkflowEngine.getMessageType

Retrieves the Workflow Engine type. Returns null for non-Workflow Engine Moolets.

Request arguments

The method takes no arguments.

Response

The method returns one of the following parameters.

Type	Description
String	One of the following: ALERT SITUATION EVENT null (if the Moolet is not a Workflow Engine)

Example

Example use of the WorkflowEngine.getMessageType method:

```
if((Bot.getType() === Bot.WORKFLOW_ENGINE)
   && (Bot.workflowEngine.getMessageType() === Bot.workflowEngine.ALERT))
   {
      logger.warning('Moolet ' + Bot.getMooletName() + ' will handle alerts')
   }
```

Process

The Process module allows you to run and control the execution of another process. You can run a process in two ways:

- Use the **`run`** method to run the process in a separate child process of Moogfarmd.
- Use the **`runToExit`** method to run the process and only return when the process exits.

Stop processes with the **`terminate`** method. These methods are detailed below.

Load the module

The Process module is available to load into any standard Moobot. To load it, define a new global variable at the top of the Moobot Javascript file. For example:

```
var proc = MooBot.loadModule('Process');
```

Method reference

The Process module uses the following methods.

create

Defines a valid pathname to an executable file that you have permission to execute (or the user that started Moogfarmd has permissions to execute).

Request arguments

The method takes the following arguments.

Name	Type	Required	Description
process	String	Yes	Path to an executable file (with permission).

Response

The method returns the following parameter.

Name	Type	Description
processObj	Object	An object containing the process to run.

arg

Defines the command line arguments required to run the process.

Request arguments

The method takes the following arguments.

Name	Type	Required
argString	Array of Strings	Yes

Response

None.

run

Takes the object returned by the **create** method and runs the process in a separate child process of Moogfarmd.

Request arguments

The method takes the following arguments.

Name	Type	Required
processObj	Object	Yes

Response

The method returns the following parameter.

Type	Description
Object	An object containing the process results.

runToExit

Takes the object returned by the **create** method, runs the process and returns when the process exits.

Request arguments

The method takes the following arguments.

Name	Type	Required
processObj	Object	Yes

Response

The method returns the following parameter.

Type	Description
Object	An object containing the process results.

terminate

Stops the process.

Request arguments

The method takes the following arguments.

Name	Type	Required
processObj	Object	Yes

Response

None.

Example

The following function runs an external tool **thisTool** using the Process module:

```
function runTool(thisTool,toolArgs,toExit)
{
var toolRun=proc.create(thisTool);
    for ( var argIdx = 0; argIdx < toolArgs.length ; argIdx++)
    {
        toolRun.arg(toolArgs[argIdx]);
    }
    if ( toExit === true )
    {
        proc.runToExit(toolRun);
        var toolResults=toolRun.output();
        toolResults=toolResults.replace("\n","");
        return(toolResults);
    }
    else
    {
        proc.run(toolRun);
        return;
    }
}
```

Usage:

```
var toolScript = "/usr/share/moogsoft/scripts/hip_chat.py";
var toolArgs = [ "--room=","Support Team", "--sigid=",sigId ];
var hipChatData = runTool( toolScript,toolArgs, true );
```

The above calls the tool runner, retrieves data, runs the process as "run to exit" (runToExit = true).

RabbitMQ

The RabbitMQ module allows you to broadcast information on a RabbitMQ bus. For example, you can use it to push alert or Situation data to a data warehouse via RabbitMQ.

You cannot connect the RabbitMQ Moobot module to the RabbitMQ instance used by Cisco Crosswork Situation Manager.

Configure the module

To use the RabbitMQ Moobot module:

- Define a new global object **`rabbit`** at the top of a Moobot JavaScript file to load the module.
- Use the **`connect`** method to create a new connection to one or more RabbitMQ brokers.
- Use the **`send`** method to send the required information.
- Use the **`close`** method to close the connection.

Refer to the Examples for more details.

Reference Guide

You can use the following methods in the RabbitMQ Moobot module.

connect

Establishes a connection to one or more RabbitMQ brokers with defined connection properties.

You cannot connect the RabbitMQ Moobot module to the RabbitMQ instance used by Cisco Crosswork Situation Manager.

Request Argument

Name	Type	Description
`<properties>`	Object	A JavaScript object containing connection properties. See below.

RabbitMQ Connection Properties

The RabbitMQ module **`connect`** method defines connection properties as a Javascript object, which may include the following keys:

Key	Description
`brokers`	Top-level container for one or more target RabbitMQ brokers. For each broker, define: **`host`**: Hostname or IP address of the RabbitMQ broker. **`port`**: Port of the RabbitMQ broker.
`user`	Username to connect to RabbitMQ.
`password`	Username to connect to RabbitMQ.
`timeout`	Length of time to wait before halting a connection or read attempt, in milliseconds. Defaults to 10,000.
`vhost`	Name of the RabbitMQ virtual host. Optional.
`ssl`	Top-level container for the SSL configuration. Optional.
`ssl_protocol`	The SSL protocol to use. If not specified, TLSv1.2 is used by default.
`server_cert_file`	Name of the SSL root CA file.
`client_cert_file`	Name of the SSL client certificate.
`client_key_file`	Name of the SSL client key file. Must be in PKCS#8 format. Refer to Message

	System SSL for more information.

Return Parameter

Type	Description
Object	A Java object containing connection details, depending on the requested connection properties. Returns **`null`** if no connection can be made.

Example

```
{
    brokers: [
    {
        host: "rabbithost",
        port: 5672
    }],
    user: "rabbitmq_admin",
    password: "78smr9!b",
    timeout: 10000,
    vhost: "rabbitvhost",
    ssl: {
        ssl_protocol: "TLSv1.2",
        server_cert_file: "server.pem",
        client_cert_file: "client.pem",
        client_key_file: "client.key"
    }
}
```

send

Sends a message to the RabbitMQ broker. Refer to the basic class in the RabbitMQ AMQP 0-9-1 Reference for a list of keys that you can specify in the message properties.

Name	Type	Description
`Exchange`	String	The RabbitMQ exchange.
`RoutingKey`	String	The RabbitMQ routing key.
`Properties`	String or Object	Message properties in one of the following formats: Plain text JSON Object payload JSON Array payload
`Message`	String	The message to send.

Return Parameter

None.

Example

```
connection.send("direct_logs", "severity",
{
    content-type : "text/xml",
```

```
    reply-to     : "greetings.hi",
    headers      : {"server" "app5.myapp.megacorp.com" "cached" false}
},
    "<Priority>1</Priority>"
)
connection.send("topic_logs", "topic", {contentType: "text/xml"},
"<Priority>1</Priority>");
```

close

Closes the connection to the RabbitMQ broker.

Request Arguments

None.

Return Parameter

Type	Description
Boolean	Indicates whether the close operation was successful.

Examples

The following examples demonstrate the use of the RabbitMQ Moobot module:

```
var rabbit = MooBot.loadModule('RabbitMQ');
var connection = rabbit.connect({
   brokers:[
      {
         host:"myHost",
         port:5672
      }
   ],
   user:"test",
   password:"test",
   timeout:10000,
   vhost:"myVHost",
   ssl:{
      ssl_protocol:"TLSv1.2",
      server_cert_file:"server.pem",
      client_cert_file:"client.pem",
      client_key_file:"client.key"
   }
});

if (connection) {
    connection.send("test", "test", {contentType: "text/xml"},
"<testKey>testValue</testKey>");
    connection.send("test", "test", {testKey: "value"});
    connection.send("test", "test", ["value"]);
    connection.send("test", "test", "testValue");
    connection.close();
}

var rabbit = MooBot.loadModule('RabbitMQ');

var connection = rabbit.connect({
   brokers:[
      {
         host:"rabbithost",
         port:5672
      }
```

```
    ],
    user:"rabbitmq_admin",
    password:"78smr9!b",
    timeout:10000,
    vhost:"rabbitvhost",
    ssl:{
       ssl_protocol:"TLSv1.2",
       server_cert_file:"server.pem",
       client_cert_file:"client.pem",
       client_key_file:"client.key"
    }
});

if (connection) {
    connection.send("testExchange", "testRoutingKey", ["one", "two"]);
    connection.close();
}
```

REST.V2

REST (Representational State Transfer) and RESTful applications use HTTP requests to post data (create and update), read data (make queries), and delete data.

The REST.V2 Moobot module accesses an external RESTful API through HTTP or HTTPS, offering consistent usage between the available methods and customization of HTTP requests sent.

It supports asynchronous operation (using callback functions) to send a request without blocking the JavaScript code execution until the request is completed. It supports use of the timeout property to make the request fail after a specified time.

REST.V2 is available to load into any standard Moobot.

To use, define a new global variable at the top of a Moobot JavaScript file to load the module. For example:

```
var REST = MooBot.loadModule('REST.V2');
```

Reference Guide

sendGet

Sends a HTTP GET request to a third party (URL) with optional parameters:

Request Arguments

Name	Type	Description
url	String	The request URL. Mandatory
<parameters>	JSON Object	Optional parameters. See below

Optional parameters

Name	Type	Description
params	String or Object	Either a String with the request encoded parameters or an Object with the parameters that will get encoded by the module.

user	String	The user name for basic authentication.
password	String	The password for basic authentication.
encrypted_password	String	Encrypted version of password (encrypted using **moog_encryptor**).
disable_certificate _validation	Boolean	'true' to disable HTTPS server certificate validation by the Moobot.
headers	Object	Any additional headers sent with the request.
callback	Callback function	The request is sent asynchronously, returns **null** and the **callback** function is called regardless of the success or failure of the request. See below.
success	Callback function	The request is sent asynchronously, returns **null** and the **success** function is called only the request was successful. See below.
failure	Callback function	The request is sent asynchronously, returns **null** and the **failure** function is called only if the request failed. See below.
timeout	Number	The period of time (in seconds) to wait for response before completing with timeout error. If 0 or less, wait indefinitely. The default is 120 seconds.
proxy	String or Object	Host, port, user, encrypted_password/password. For example, as an object: **proxy:{** **host:"proxyhost",** **port:1223,** **user:"proxyuser",** **encrypted_password:"2KctaEbJH/m8rz4WqgmZYZfdripdIsku7fOFJWM6YNA="** **//password: "unencrypted_plain_text_password"** **}** As an object, you can either specify a Cisco encrypted password or a plain text password, specifying both will favour the encrypted_password value. Or, as a string, where format is <user>:<password>@<host>:<port> **proxy: "proxyuser:passw0rd@proxyhost:1223"** Only plain text passwords are supported in the string format.

Sending an asynchronous request (with callback functions)

To send a request without blocking the javascript code execution until the request is completed, define one (or more) of the callback functions: **`callback`**, **`success`** and **`failure`**. The REST.V2 module method (**`send...`**) then returns **`null`**, and sends the request in another thread.

Return Parameters

Sending a synchronous request returns a JavaScript object with the following fields:

Name	Type	Description
`success`	Boolean	True if and only if the request was successful
`status_code`	Number	The HTTP status code of the request (200 = OK, 404 = Not found. Full list at w3.org)
`status_msg`	String	The message from the request ("OK", "Not found", etc.)
`response`	String	The response as raw text Currently, binary response is not supported
`headers`	Object	The response HTTP headers

Sending an asynchronous request (with callback functions) returns **`null`**. Once the request has completed, the callback function(s) are called with the reply object as the first (optional) parameter and the request object as the second (optional) parameter.

Examples

Each of the following gives details on the Cisco home page:

Synchronous request

```
var rc = REST.sendGet('http://www.Cisco.com');
```

Asynchronous request

```
function restSuccess(rc)
{
    var response = JSON.parse(rc.response);
    logger.info("number = " + response.records[0].number);
}

function restFailed(rc, req)
{
    var response = JSON.parse(rc.response);
    logger.info("URL:" + req.url +" failed - Msg:" + response.status_msg);
}

REST.sendGet({url: "http://www.Cisco.com",
                          success: restSuccess,
                          failure: restFailed});
```

Response

```
{
    "status_code": 200,
    "success": true,
```

```
        "response": "<!DOCTYPE html>... </body></html>",
        "status_msg": "OK",
        "headers": {
            "Transfer-Encoding": [
                "chunked"
            ],
            "Keep-Alive": [
                "timeout=15, max=100"
            ],
            "Server": [
                "Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.10 with Suhosin-Patch
    mod_ssl/2.2.22 OpenSSL/1.0.1"
            ],
            "Connection": [
                "Keep-Alive"
            ],
            "Vary": [
                "Accept-Encoding"
            ],
            "Date": [
                "Fri, 30 Jan 2015 12:37:13 GMT"
            ],
            "Content-Type": [
                "text/html"
            ],
            "X-Powered-By": [
                "PHP/5.3.10-1ubuntu3.10"
            ]
        }
    }
```


sendPost

Sends a HTTP POST request to a third party (URL) with optional parameters:

Request Arguments

Name	Type	Description
url	String	The request URL. Mandatory
<parameters>	JSON Object	Optional parameters. See below

Optional parameters

Name	Type	Description
params	String or Object	Either a string with the request encoded parameters or an object with the parameters that will get encoded by the module.
content_type	String	The content type of the body.
body	String or Object	The request body. Either a string (that will be sent as is) or an object. If the **content_type** is “application/json” and the body is an object, the body will be sent as JSON. Otherwise it will be sent as URL encoded.
user	String	The user name for basic authentication.
password	String	The password for basic authentication.
encrypted_passwo	String	Encrypted version of password (encrypted using **moog_encryptor**).

rd		
disable_certificate_validation	Boolean	'true' to disable HTTPS server certificate validation by the Moobot.
headers	Object	Any additional headers sent with the request.
callback	Callback function	The request is sent asynchronously, returns **null** and the **callback** function is called regardless of the success or failure of the request. See below.
success	Callback function	The request is sent asynchronously, returns **null** and the **success** function is called only the request was successful. See below.
failure	Callback function	The request is sent asynchronously, returns **null** and the **failure** function is called only if the request failed. See below.
timeout	Number	The period of time (in seconds) to wait for response before completing with timeout error. If 0 or less, wait indefinitely. The default is 120 seconds.
proxy	String or Object	Host, port, user, encrypted_password/password. For example, as an object: **proxy:{** **host:"proxyhost",** **port:1223,** **user:"proxyuser",** **encrypted_password:"2KctaEbJH/m8rz4WqgmZYZfdripdIsku7fOFJWM6YNA="** **//password: "unencrypted_plain_text_password"** **}** As an object, you can either specify a Cisco encrypted password or a plain text password, specifying both will favour the encrypted_password value. Or, as a string, where format is <user>:<password>@<host>:<port> **proxy: "proxyuser:passw0rd@proxyhost:1223"** Only plain text passwords are supported in the string format.

Sending an asynchronous request (with callback functions)

To send a request without blocking the JavaScript code execution until the request is completed, define one (or more) of the callback functions: **callback**, **success** and **failure**. The REST.V2 module method (**send...**) then returns **null**, and sends the request in another thread.

Return Parameters

Sending an asynchronous request (with Callback functions) returns **`null`**. See above.

Sending a synchronous request returns a JavaScript object with the following fields:

Name	Type	Description
`success`	Boolean	True if and only if the request was successful.
`status_code`	Number	The HTTP status code of the request. (200 = OK, 404 = Not found. Full list at w3.org)
`status_msg`	String	The message from the request ("OK", "Not found", etc.)
`response`	String	The response as raw text. Binary response is not supported.
`headers`	Object	The response HTTP headers.

Sending an asynchronous request (with callback functions) returns **`null`**. Once the request has completed, the callback function(s) are called with the reply object as the first (optional) parameter and the request object as the second (optional) parameter.

Examples

Each of the following accesses DuckDuckGo and searches for 'Cisco'.

Synchronous request:

```
var rc = REST.sendPost('https://api.duckduckgo.com/',
{q:'Cisco', format:'json', pretty:1});
```

Asynchronous request:

```
REST.sendPost({url: 'https://api.duckduckgo.com/',
   body: {q:'Cisco', format:'json', pretty:1},
   timeout: 4.2,
   callback: function(rc) {
   ...
   }});
```

Here, the request has a timeout set of 4.2 seconds.

Responses

For the synchronous request, and for the asynchronous request if it doesn't time out:

```
{
    "status_code": 200,
    "success": true,
    "response": "{   \"DefinitionSource\" : \"\",   \"Heading\" : \"\",
\"ImageWidth\" : 0,   ... : \"\"}",
    "status_msg": "OK",
    "headers": {
        "Transfer-Encoding": [
            "chunked"
        ],
        "Strict-Transport-Security": [
            "max-age=0"
        ],
        "Cache-Control": [
            "max-age=1"
```

```
            ],
            "Server": [
                "nginx"
            ],
            "X-DuckDuckGo-Results": [
                "1"
            ],
            "X-DuckDuckGo-Locale": [
                "en_US"
            ],
            "Connection": [
                "keep-alive"
            ],
            "Expires": [
                "Fri, 30 Jan 2015 12:44:47 GMT"
            ],
            "Date": [
                "Fri, 30 Jan 2015 12:44:46 GMT"
            ],
            "Content-Type": [
                "application/x-javascript"
            ]
        }
    }
```

...if the asynchronous request times out:

```
    {
        "status_code": 408,
        "success": false,
        "status_msg": "Request Time-Out"
    }
```

sendPut

Sends a HTTP PUT request to a third party (URL) with optional parameters:

Request Arguments

Name	Type	Description
url	String	The request URL. Mandatory.
<parameters>	JSON Object	Optional parameters. See below.

Optional parameters

Name	Type	Description
params	String or Object	Either a string with the request encoded parameters or an object with the parameters that will get encoded by the module.
content_type	String	The content type of the body.
body	String or Object	The request body. Either a string (that will be sent as is) or an object. If the **content_type** is "application/json" and the body is an object, the body will be sent as JSON. Otherwise it will be sent as URL encoded.

user	String	The user name for basic authentication.
password	String	The password for basic authentication.
encrypted_password	String	Encrypted version of password (encrypted using **moog_encryptor**).
disable_certificate_validation	Boolean	'true' to disable HTTPS server certificate validation by the Moobot.
headers	Object	Any additional headers sent with the request.
callback	Callback function	The request is sent asynchronously, returns **null** and the **callback** function is called regardless of the success or failure of the request. See below.
success	Callback function	The request is sent asynchronously, returns **null** and the **success** function is called only the request was successful. See below.
failure	Callback function	The request is sent asynchronously, returns **null** and the **failure** function is called only if the request failed. See below.
timeout	Number	The period of time (in seconds) to wait for response before completing with timeout error. If 0 or less, wait indefinitely. The default is 120 seconds.

Sending an asynchronous request (with callback functions)

To send a request without blocking the JavaScript code execution until the request is completed, define one (or more) of the callback functions: **callback**, **success** and **failure**. The REST.V2 module method (**send...**) then returns **null**, and sends the request in another thread.

Return Parameters

Sending a synchronous request returns a JavaScript object with the following fields:

Name	Type	Description
success	Boolean	True if and only if the request was successful.
status_code	Number	The HTTP status code of the request. (200 = OK, 404 = Not found. Full list at w3.org)
status_msg	String	The message from the request ("OK", "Not found", etc.)
response	String	The response as raw text. Binary response is not supported.
headers	Object	The response HTTP headers.

Sending an asynchronous request (with callback functions) returns **null**. Once the request has completed, the callback function(s) are called with the reply object as the first (optional) parameter and the request object as the second (optional) parameter.

Example

The following stores the specified information at the URL (similar to a file upload):

Request

```
var rc = REST.sendPut('http://api.acme.com/reportIncident', '{"incident":"broken
fan","location":"office2"}');
```

Response

```
{
    "status_code": 204,
    "success": true,
    "response": "",
    "status_msg": "No Content",
    "headers": {
        "Connection": [
            "keep-alive"
        ],
        "Date": [
            "Fri, 30 Jan 2015 12:55:59 GMT"
        ]
    }
}
```

When POSTing or PUTting URL encoded data (a content-type of "application/x-www-form-urlencoded") complex objects will need to be either split into individual key:value pairs suitable for url encoding or simply JSON stringify the object in its entirety. Stringifying the object will require the receiver to be able to parse the string value back to an object if needed. If the receiver cannot do this parsing then the object will need to be broken into key value pairs. For example, to send the entire alert custom_info object as part of a URL-encoded body:

```
var custom_info = alert.getCustomInfo();
var payload;
try {
payload = JSON.stringify(custom_info);
}
catch(e) {
logger.info("Failed to stringify custom_info " + e );
payload = null;
}


        var postParams={
            "url" : "http://www.someurl.com/someEndpoint",
            "body" : payload,
            "content_type" : "application/x-www-form-urlencoded"
};

var request = rest.sendPost(postParams);
```

sendDelete

Sends an HTTP DELETE request to a third party (URL) with optional parameters:

Request Arguments

Name	Type	Description
url	String	The request URL. Mandatory.
<parameters>	JSON Object	Optional parameters. See below.

Optional parameters

Name	Type	Description
params	String or Object	Either a string with the request encoded parameters or an object with the parameters that will get encoded by the module.
user	String	The user name for basic authentication.
password	String	The password for basic authentication.
encrypted_password	String	Encrypted version of password (encrypted using **moog_encryptor**).
disable_certificate_validation	Boolean	'true' to disable HTTPS server certificate validation by the Moobot.
headers	Object	Any additional headers sent with the request.
callback	Callback function	The request is sent asynchronously, returns **null** and the **callback** function is called regardless of the success or failure of the request. See below .
success	Callback function	The request is sent asynchronously, returns **null** and the **success** function is called only the request was successful. See below.
failure	Callback function	The request is sent asynchronously, returns **null** and the **failure** function is called only if the request failed. See below.
timeout	Number	The period of time (in seconds) to wait for response before completing with timeout error. If 0 or less, wait indefinitely. The default is 120 seconds.

Sending an asynchronous request (with callback functions)

To send a request without blocking the javascript code execution until the request is completed, define one (or more) of the callback functions: **callback**, **success** and **failure**. The REST.V2 module method (**send...**) then returns **null**, and sends the request in another thread.

Return Parameters

Sending a synchronous request returns a JavaScript object with the following fields:

Name	Type	Description
success	Boolean	True if and only if the request was successful.
status_code	Number	The HTTP status code of the request. (200 = OK, 404 = Not found. Full list at w3.org)
status_msg	String	The message from the request ("OK", "Not found", etc.)
response	String	The response as raw text. Binary response is not supported.
headers	Object	The response HTTP headers.

Sending an asynchronous request (with callback functions) returns **`null`**. Once the request has completed, the callback function(s) are called with the reply object as the first (optional) parameter and the request object as the second (optional) parameter.

Example

The following sends a delete request to the specified URL, with additional headers criteria:

Request:

```
var rc =
REST.sendDelete({url:"http://moogbox2:9090/deletePassport/123456789","headers":{
"user-agent":"moobot","accept":"text/plain","accept-language":"en-US"}});;
```

Response

```
{
    "status_code": 200,
    "success": true,
    "response": "{\t\"remoteId\": 33,\t\"weight\":
0.8240487528964877,\t\"location\": {\t\t\"latitude\":
147.3387699946761,\t\t\"longitude\": -7.957067163661122\t}}",
    "status_msg": "OK",
    "headers": {
        "Transfer-Encoding": [
            "chunked"
        ],
        "Connection": [
            "keep-alive"
        ],
        "Date": [
            "Fri, 30 Jan 2015 12:49:44 GMT"
        ],
        "Content-Type": [
            "application/json"
        ]
    }
}
```

send

A generic send request for sending other HTTP methods as part of the request properties (**`'GET'`**, **`'HEAD',`** etc.). Optional parameters for synchronous and asynchronous requests are available as described in the above methods.

Example

The following returns time/date information from the Cisco server:

Request

```
var rc = REST.send({method: 'HEAD', url: 'http://www.Cisco.com/'});
logger.warning("rc: " + JSON.stringify(rc, null, "\t") );
var date = rc.headers.Date[0];
logger.warning("date " + date );
```

Response

```
{
    "status_code": 204,
    "success": true,
    "response": "",
    "status_msg": "OK",
    "headers": {
        "Keep-Alive": [
            "timeout=15, max=100"
        ],
        "Server": [
            "Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.10 with Suhosin-Patch
mod_ssl/2.2.22 OpenSSL/1.0.1"
        ],
        "Connection": [
            "Keep-Alive"
        ],
        "Vary": [
            "Accept-Encoding"
        ],
        "Date": [
            "Fri, 30 Jan 2015 13:00:33 GMT"
        ],
        "Content-Type": [
            "text/html"
        ],
        "X-Powered-By": [
            "PHP/5.3.10-1ubuntu3.10"
        ]
    }
}
```

Proxy examples

The following examples show how to configure Cisco Crosswork Situation Manager Moobots when a proxy server is used for connection to Cisco.

You can define a proxy in the following ways:

```
proxy: "proxyuser:passw0rd@proxyhost:1223"

proxy: "proxyhost:1223"

proxy: {
        host: "proxyhost",
        port: 1223
        }
```

Situation Manager

The following example shows how to update the Situation Manager to send a REST.V2 updateSituation message through a proxy server.

- Edit the Situation Manager Moobot file, located at**`$MOOGSOFT_HOME/bots/moobots/SituationMgr.js`**.
- Modify the**`updateSitn`**function to utilize the POST action. For example:

```
function updateSitn(situation)
{
        var sig_id = situation.value("sig_id");
        logger.warning("Update Situation Processed: " + sig_id);
```

```
    doPOST(sig_id);
}
```

- Insert the proxy block into the **POST** action. For example:

```
function doPOST(sig_id)
{
        var request = REST.sendPost({
                url:"http://surveilanceserver_84:9090/reportAntiSoc",
                params: {
                        crime: "Graffiti"
                },
        proxy: {
            host: "proxyserver",
            port : 3128,
            user : "username",
            encrypted_password:"zm0lxjTGiAhp6LrpM49+kr4SDtHj/fq16+i+hD1MG4c="
        },
                callback: function(response, request)
                {
                        if (response.success) {
                                logger.warning("4764 CALLBACK SUCCESS
("+sig_id+") RESPONSE - ("+response.status_code +" - "+response.response+")
REQUEST - "+ JSON.stringify(request));
                        } else {
                                logger.warning("4764 CALLBACK FAILURE
("+sig_id+") RESPONSE - ("+response.status_code +" - "+response.response+" -
"+response.status_msg+") REQUEST -
                                " + request.status_code + " " +
request.response + " " + request.status_msg);
                        }
                }
        });
        logger.warning("4764 POST REQUEST SENT FOR "+sig_id+" ...");
}
```

ServiceNow

The following example demonstrates how to configure the ServiceNow ticketing integration when Cisco Crosswork Situation Manager is installed on-prem and ServiceNow is in the cloud, and the two systems communicate through a proxy server.

Edit the ServiceNow Moobot file, located at `$MOOGSOFT_HOME/bots/moobots/ServiceNow-2.0-Geneva.js` and define a variable containing the proxy details. For example:

```
var proxy = {
  host: 'proxy-app.company.com',
  timeout:60,
  port: 8080
  }
```

- Add the proxy to the POST actions in the **`AddToWorkNotes`** and **`resolveIncident`** functions. For example:

```
var rc = REST.sendPost({
  'url': url,
  'body': JSON.stringify(urlParameters),
  'user': user,
```

```
    'password': password,
    'content_type': "application/json",
    'proxy': proxy,
    'disable_certificate_validation': true
    });
```

Utilities

The Utilities module is a JavaScript utility that allows you to escape XML so that Cisco Crosswork Situation Manager correctly interprets control characters as data, not markup.

You can also use the module to convert an XML string to a JSON object, which is easier to manipulate in JavaScript. You can convert a JSON object to XML for external communication that requires XML input.

Load the module

The Utilities module is available to load into any standard Moobot or LAMbot. To load it, define a new global variable at the top of the Moobot or LAMbot Javascript file. For example:

```
var utilities = MooBot.loadModule('Utilities');
```

```
var utilities = LamBot.loadModule('Utilities');
```

Method reference

The Utilities module uses the following methods.

escapeXML

Escapes an XML string. Certain characters will not parse correctly if they are not escaped:

Unescaped character	Escaped string
"	**"**
'	**'**
<	**<**
>	**>**
&	**&**

Request arguments

The method takes the following arguments:

Name	Type	Required
value	String	Yes

Example

Example use of the **escapeXML** method:

```
var unescapedXML = 'my content requires "< and > "';
var escapedXML = '<tag>' + utilities.escapeXML(unescapedXML) + '</tag>';
```

The variable **escapedXML** now contains:

```
<tag>my content requires &amp;quot;&amp;lt; and &amp;gt; &amp;quot;</tag>
```

unescapeXML

Unescapes an XML string.

Request arguments

The method takes the following arguments:

Name	Type	Required
value	String	Yes

Example

Example use of the **unescapeXML** method:

```
var escapedXML = '<tag>my content requires &quot;&lt; and &gt; &quot;</tag>';
var unescapedXML = utilities.unescapeXML(escapedXML);
```

The variable **unescapedXML** now contains:

```
<tag>my content requires "< and > "</tag>
```

xmlToJSON

Converts an XML string to a JSON object.

Request arguments

The method takes the following arguments:

Name	Type	Required
value	XML string	Yes

Example

Example use of the **xmlToJSON** method:

```
var xmlExample = '<alerts>' +
    '<alert enriched="false">' +
    '<id>1</id>' +
    '<description>Alert 1</description>' +
    '<host>email.moogsoft.com</host>' +
    '<severity>5</severity>' +
    '</alert>' +
    '<alert enriched="true">' +
    '<id>2</id>' +
    '<description>Alert 2</description>' +
    '<host>calendar.moogsoft.com</host>' +
    '<severity>2</severity>' +
    '</alert>' +
    '</alerts>';

var alerts = utilities.xmlToJSON(xmlExample);
```

The variable **alerts** now contains:

```
{
  "alerts":{
    "alert":
    [
```

```
        {
          "severity":5,
          "host":"email.moogsoft.com",
          "description":"Alert 1",
          "id":1,
          "enriched":false
        },
        {
          "severity":2,
          "host":"calendar.moogsoft.com",
          "description":"Alert 2",
          "id":2,
          "enriched":true
        }
      ]
    }
  }
```

jsonToXML

Converts a JSON object to an XML string. You can only use the utility to convert JSON objects, not arrays.

Request arguments

The method takes the following arguments:

Name	Type	Required
value	JSON object	Yes

Example

Example use of the **jsonToXML** method:

```
var jsonObjectExample =
{
  "data": {
    "alerts":
    [
      {
         "enriched": "false",
         "id": "1",
         "description": "Alert 1",
         "host": "email.moogsoft.com",
         "severity": "5"
      },
      {
         "enriched": "true",
         "id": "2",
         "description": "Alert 2",
         "host": "calendar.moogsoft.com",
         "severity": "2"
      }
    ]
  }
};

var convertedXML = utilities.jsonToXML(jsonObjectExample);
```

The variable **convertedXML** now contains:

```
<data>
   <alerts>
      <severity>5</severity>
      <enriched>false</enriched>
      <host>email.moogsoft.com</host>
      <description>Alert 1</description>
      <id>1</id>
   </alerts>
   <alerts>
      <severity>2</severity>
      <enriched>true</enriched>
      <host>calendar.moogsoft.com</host>
      <description>Alert 2</description>
      <id>2</id>
   </alerts>
</data>
```

Alert Builder Reference

This is a reference for the Alert Builder Moolet.Alert Builder

You can change the behavior of the Alert Builder by editing the configuration properties in the **$MOOGSOFT_HOME/config/moolets/alert_builder.conf** configuration file. It contains the following properties:

name

Name of the Alert Builder Moolet. Do not change.

Type	String
Required	Yes
Default	**"AlertBuilder"**

classname

Moolet class name. Do not change.

Type	String
Required	Yes
Default	**4"CAlertBuilder"**

run_on_startup

Determines whether the Alert Builder runs when Cisco Crosswork Situation Manager starts. By default, it is set to true, so that when Moogfarmd starts, it automatically creates an instance of the Alert Builder. In this case you can stop it using **farmd_ctrl**.

Type	Boolean
Required	Yes
Default	**true**

moobot

Specifies a JavaScript file found in **`$MOOGSOFT_HOME/moobots`**, which defines the Alert Builder Moobot, which creates alerts.

Type	String
Required	Yes
Default	**`AlertBuilder.js`**

metric_path_moolet

Determines whether or not Cisco Crosswork Situation Manager includes the Alert Builder in the Event Processing metric for Self Monitoring.

Type	Boolean
Required	Yes
Default	**`true`**

event_streams

A list of event streams, which the Alert Builder Moolet processes in this instance of Moogfarmd. The LAMs can be configured to send events on different streams. Moogfarmd, as specified in the Alert Builder configuration, then decides whether or not to process them. If Cisco Crosswork Situation Manager runs multiple Moogfarmds, you can have different event streams being processed by different Alert Builder Moolets.

You can comment out **`event_streams`**, or provide an empty list. Then, the Alert Builder processes every event that is published on the default **`/Events`** topic on the Message Bus.

You configure the Alert Builder Moolet by giving it a list of strings, for example, [“App A”, “App B”]. The result is that the Alert Builder listens for events published on **`/Events/AppA`**, and **`/Events/AppB`**, and processes that data. Importantly, in this example, events published to **`/Events`** or any other stream are ignored. You can have Moogfarmds that process completely separate event streams, or, multiple Moogfarmds that process some different event streams and some common event streams. You would do this when some of the alerts are common to all the applications that are being processed, but some are specific only to a given application. In this way, you can cluster alerts separately for each application by configuring the Sigalisers to only processes alerts from a specific upstream Alert Builder Moolet.

For example, if you have two separate applications that share the same network infrastructure: in Moogfarmd 1, you can have as the event streams, application A and networks, and, in Moogfarmd 2, you can have application B and networks. With this configuration, you can detect alerts and then create Situations that are relevant for just application A and similarly just for application B; however, if there is common networking infrastructure and problems occur with network failures across applications A and B, the Alert Builder can cluster these into Situations.

Type	String
Required	No
Default	**`[ "AppA" ]`**

threads

Specifies the number of threads in the Alert Builder. Choose a value to match the event rate experienced by your system that allows time for alert creation.

Type	String
Required	Yes
Default	**4**

events_analyser_config

Allows you to specify a different Events Analyser configuration, for tokenizing and analysis rules, for each Alert Builder Moolet. If no configuration file is specified, the system default **`events_analyser.conf`** is used.

Type	String
Required	No
Default	**`"events_analyser.conf"`**

priming_stream_name

Stream name under which the Events Analyser runs in order to calculate token and alert entropies. If set to **`null`**, all alerts from all streams are included in the entropy calculations.

Type	String
Required	Yes
Default	**`null`**

priming_stream_from_topic

If set to **`true`**, Moogfarmd extracts the priming stream name from the event's stream. If set to **`false`**, Moogfarmd uses the stream configured in **`priming_stream_name`**.

Type	Boolean
Required	Yes
Default	**`false`**

Alert and Event Field Reference

This is a reference guide for alert and event fields, input types, field descriptions and output examples.

Field	Type	Description	Example Output
`active_situations`	Array	IDs of any Situations associated with the alert.	1, 6, 8
`agent_host`	Text	Host machine or physical location of the agent that created the event.	OEM Monitor 1
`agent_name`	Text	Name of the agent that created the event.	NAGIOS SOCKET

agent_location	Text	Host machine or physical location of the agent that created the event.	London Data Centre (51.4167,-0.2833)
agent_time	Integer	Timestamp when the event occurred in epoch time. Use **$moog_now** in the mapping to set agent time to the time the event arrived at Cisco Crosswork Situation Manager.	1516183437
alert_id	Integer	Internal identifier generated by Cisco Crosswork Situation Manager.	101
class	Text	Level of classification for an event. This follows the hierarchy; **class** then **type**.	CISCO-IF-Extension-MIB
count	Integer	Number of events in the alert.	2
custom_info	Text	Custom information added as a JSON encoded string.	**custom_info.myNodeList=["node1" , "node2" , "node3"]**
description	Text	Text description of the alert.	Network Interface (ifIndex = 512479388) Up (ifEntry.52683483)
entropy	Integer	Measure of uncertainty of an outcome between 0 and 1 (0 meaning very certain and 1 meaning very uncertain).	0.4
external_id	Integer	Unique identifier from the event source.	7622183
first_event_time	Integer	Earliest event time for the alert. This is calculated from the **agent_time** of the events that constitute the alert.	14:08:14 16/01/2018
host	Text	Name of the source machine that generated the event.	OEM Server 2
internal_last_event_time	Integer	Time that the latest event for the alert was received by the Cisco Crosswork Situation Manager server.	10:24:03 19/01/2018
last_change	Integer	Time that the alert was last updated in the Cisco	12:38:06 19/01/2018

		Crosswork Situation Manager UI.	
last_event_time	Integer	Latest event time for the alert. This is calculated from the **agent_time** of the events that constitute the alert.	10:24:03 19/01/2018
manager	Text	General identifier of the event generator or intermediary.	NAGIOS, SCOM.
owned_by	Text	Alert owner's username.	John Smith
severity	Integer	Severity level of the alert between 0 and 5.	4
significance	Integer	Relative Significance of an alert is calculated based on its entropy.	3
situations	Array	Any Situations the alert is associated with, including those that have been resolved or closed.	24, 01
source	Text	Name of the source machine that generated the event. If there is no source machine or application, the source is the name of the instance (database name, cluster node, container name).	A hostname or fully qualified domain name (FQDN).
source_id	Text	Identifier for the source machine that generated the event.	5dc68d65-532c-4918-be12-21e1cbcf7af2
status	Text	Status of the alert.	Assigned
type	Text	Level of classification for an event. This follows the hierarchy; **class** then **type**.	CISCO-IF-Extension-MIB Notification

Event and Alert Field Best Practice

This best practice is an attempt to offer consistency and reuse of configurations including the mapping from a source to an inbound event. The fields exposed in the alert/event are:

	Field	Requi	Data Type	Size	Description	Example	Comment

		red					
1	signature	Yes	VARBINARY(binary)	767	This is a special attribute used to determine when Cisco Crosswork Situation Manager deduplicates events into Alerts. It can be any combination of one or more of the attributes listed below To be constructed as a subset of events from a source, also see existing guidance Constructed fields should be separated by "::" avoiding any possible issues with concatenation providing misleading results. e.g. NodeA event id 12 would concatenate as NodeA12, which would be the same as NodeA1 event 2. NodeA::12 and NodeA1::2 would therefore differentiate Signatures do not need to be human readable, so clarity isn't a concern. If length is becoming an issue - remove whitespace or other extraneous characters (via a lambot)	host1::nagios::cpu	
2	alert_id	Yes	BIGINT(binary)	20	An auto-assigned incremental number. Internally generated DO NOT CHANGE		
3	source_id	Yes	TEXT(utf8)	65535	Source and Source_ID refer to the generating source of the event, primarily focused on the host environment. The Source should be any unique human readable name (FQDN, Hostname, etc) and	192.168.1.107	

					the source_id should be any identifier for the source machine generated (IP, MAC, CI Number, etc.) If the event has no machine identification such as Application or other software generated events, then the Source should be some unique identifier of the instance (database name, cluster node, container name etc.). Again source_id should be any other unique identifier that is available (container UUID, cluster node UUID etc.) This attribute can be used for any additional identification attribute of the CI		
4	external_id	No	TEXT(utf8)	65535	Any unique identifier provided in the source event (event ID, Incident ID etc.) This is typically set to the CI's ID in the CMDB, or where the event is emitted from an underlying element management system, and may hold the unique source event identifier	12345	Returns Null if blank
5	manager	No	TEXT(utf8)	65535	A general identifier of the event generator or intermediary (NAGIOS, SCOM, etc.) In hub-and-spoke and/or relay architectures this typically is the name of	Nagios	Returns Null if blank

					the agent manager that pre-aggregates events prior to sending to Cisco Crosswork Situation Manager. For example, there may be an BMC Patrol manager that manages all San Francisco data center alerts. This field is also sometimes used simply to track the name of the Cisco Crosswork Situation Manager LAM that received the alerts in multi-LAM deployments		
6	source	Yes	TEXT(utf8)	65535	Source and Source ID refer to the generating source of the event, primarily focused on the host environment. The source should be any unique human readable name (FQDN, Hostname, etc) and the source_id should be any identifier for the source machine generated (IP, MAC, CI Number, etc.) If the event has no machine identification such as Application or other software generated events, then the Source should be some unique identifier of the instance (database name, cluster node, container name etc.). Again source_id should be any other unique identifier that is available (container UUID, cluster node UUID etc.)	host1	
7	class	Yes	TEXT(utf8)	65535	Class and Type are generic classifications for the event in a hierarchy that allow	cpu	

					you to maintain a simple event ontologies; class then type. (Disk space: free space, Memory: max used...total available, etc.)		
8	agent	Yes	TEXT(utf8)	65535	The specific agent that created the event, (SCOM REST, NAGIOS SOCKET, SNMP TRAP NATIVE, etc.). This is typically the name of the agent that facilitates the event from the CI e.g. "nagios-agent-london-7" A simple way to provide this is in the lam.conf by setting the agent:name and then mapping $LamInstanceName to agent, *this is the default* **{ name: "agent",rule: "$LamInstanceName" },**	Linux	
9	agent_location	Yes	TEXT(utf8)	65535	This is typically the geographic location of the agent and/or CI such as "London". Should be used consistently for all sources, either as the host machine that the agent is executed from (BEM Server 1, OEM Monitor cluster, etc.) OR the physical location that the agent is executing (NYC Data Centre, Stuttgart Main Station, (51.407139, -0.307321) etc.)	New York, NY	

10	agent_time	Yes			This is the timestamp in epoch seconds when the event occurred. This should be set across all event sources to provide a common time reference. Timezones should be nullified - all events should be presented in the same time context. If an event source does not provide a suitable time in the payload then use the ingestion time at the LAM. Note: polled event sources (rest_client_lam, SCOM, Netcool) may skew the event time in line with the poll cycle. If an event is being generated in a different timezone and is manipulated into the Cisco Crosswork Situation Manager server time - add the origin time to the custom_info for the event. This can be operationally useful. e.g. custom_info.originalEventTime : agent_time should be in epoch seconds - convert as necessary. Miscalculated event times will cause unpredictable results across the system. Also see 4.1.2 Release note. [MOOG-2278] - Enhanced Alert Times If the agent_time is not defined, it should be set to the current epoch time using Javascript functions such as:		

					Math.round(Date.now() / 1000);		
11	type	Yes	TEXT(utf8)	65535	Class and Type are generic classifications for the event in a hierarchy that allow you to maintain a simple event ontologies; class then type. (Disk space: free space, Memory: max used...total available, etc.)	DOWN	
12	severity	Yes	INT(binary)	11	Standard 0-5 but be mindful of the significance across all event sources if possible. A low value event source could produce critical events that in the wider context would be considered minor Use the Cisco Crosswork Situation Manager LAM config file built in "sevMapper" to map your incoming severity values to a number between 0 and 5 : 0 = Clear 1 = Indeterminate 2 = Warning 3 = Minor 4 = Major 5 = Critical	5	0 clear - 5 critical
13	significance	No	INT(binary)	11	This value is calculated by Cisco Crosswork Situation Manager Events Analyser. Internally generated		

					DO NOT CHANGE		
14	count	No	INT(binary)	11	The reference count of deduplicated Events for each Alert. Internally generated DO NOT CHANGE		
15	description	Yes	TEXT(utf8)	65535	The main text payload of the event. Add as much textual detail as possible. Remember a human operator will look at the detail and the entropy calculation works best with detailed narratives.	CPU Threshold exceeded: 99%	
16	first_event_time	No	BIGINT(binary)	20	If you set agent_time in the LAM/LAMbot to the actual epoch seconds timestamp of each event, Cisco Crosswork Situation Manager will automatically keep track of the first and last occurrence of multiple instances of the same event. Internally generated DO NOT CHANGE		
17	last_event_time	No	BIGINT(binary)	20			
18	int_last_event_time	No	BIGINT(binary)	20	Internally generated DO NOT CHANGE	1411134582	Fromagent_time
19	last_state_change	No	BIGINT(binary)	20	Internally generated DO NOT CHANGE		
20	state	No	INT(binary)	11	1 | Opened 2 | Unassigned 3 | Assigned 4 | Acknowledged 5 | Unacknowledged 6 | Active		

					7 | Dormant 8 | Resolved 9 | Closed 10 | SLA Exceeded Internally generated DO NOT CHANGE		
21	owner	No	INT(binary)	11	Set when an operator right-clicks on an alert in the Cisco Crosswork Situation Manager UI and assigns ownership. Internally generated DO NOT CHANGE		
22	entropy	No	DOUBLE(binary)	22	Internally generated DO NOT CHANGE		
23	custom_info	No	TEXT(utf8)	65535	Custom_info is a special field that is the mechanism for extending the Cisco Crosswork Situation Manager alert schema. This is a JSON encoded string that should contain key value pairs for each data element not supplied in the initial event or having been enriched via alert transformation. Be consistent with key names so they can be used in Sigalisers and filters. Consider using a LAMBot module that sets a base set of custom_info across all lams - this provides a single point of administration for the customer. Care should be taken when setting custom_info in a LAM to ensure that it does		Returns Null if blank

					not overwrite downstream additions (e.g. enrichment via a moobot) when the Event is de-duplicated. You can store simple or arbitrarily complex hierarchical JSON attributes in this field. They are basically serialized for use in the standard JSON.parse/stringify manner and Cisco Crosswork Situation Manager UI is written to display JSON hierarchies of any complexity in a tree-view format		

Moolets

A Moolet is an intelligence module that is used to perform specific services in Cisco Crosswork Situation Manager. Some Moolets have an accompanying Moobot, a Javascript file that controls or customizes Moolet behavior.

Events can trigger a Moolet in Moogfarmd as follows:

1. process_output_of: The Moolet listens for events from another named Moolet. You can use this method to chain Moolets together to form an automated workflow pipeline.
2. mooms_event_handler: The Moolet listens for events on the Message Bus, for example actions triggered by a user or within another instance of moogfarmd.
3. standalone_moolet: The Moolet listens for events generated by other Moolets within the same Moogfarmd instance without being part of the same process chain.
4. scheduler: A unique Moolet type that allows time based task execution.

Refer to the documentation on individual Moolets to learn about how to configure their behavior:

Configure Alert Behavior During a Maintenance Window

The Maintenance Window Manager Moolet compares alerts against active maintenance windows. If the alerts match an active Maintenance Schedule filter, then they are not forwarded onto the next part of the chain. This prevents a Sigaliser Moolet clustering these alerts into Situations.

To schedule a maintenance window, see Schedule Maintenance Downtime.

Configure Maintenance Window Manager

Edit the configuration file at **`$MOOGSOFT_HOME/config/moolets/maintenance_window_manager.conf`**.

Refer to Maintenance Window Manager Reference to see all available properties.

Example configuration

The following example demonstrates a simple Maintenance Window Manager configuration:

```
{
   name                     : "MaintenanceWindowManager",
   classname                : "CMaintenance",
   run_on_startup           : true,
   metric_path_moolet       : true,
   process_output_of        : "AlertBuilder",
   maintenance_status_field : "maintenance_status",
   maintenance_status_label : "In maintenance",
   update_captured_alerts   : true
}
```

Maintenance windows

You can use the Maintenance Schedule functionality to schedule outages when you do not want new Situations to be created from these alerts. You can configure the Maintenance Manager Moolet to ensure that alerts are not passed along to Sigalisers and clustered into Situations during that time period. You can set up maintenance windows using:

- UI: See Schedule Maintenance Downtime for more information on how to set up maintenance windows.Schedule Maintenance Downtime
- Graze API.

Updating captured alerts

In addition to implementing the maintenance windows, the Maintenance Window Manager Moolet updates the following **`custom_info`** fields in each alert affected by a maintenance window. Because the Maintenance Window Manager uses these **`custom_info`** fields within the alerts, Moobots must not overwrite these **`custom_info`** fields or completely empty the **`custom_info`** object within alerts.

Field	Description
custom_info.maintenance_status	Configurable text label. Set to **`"In maintenance"`** by default.
custom_info.maintenance_id	Numerical ID of the maintenance window that captured the alert.
custom_info.maintenance_name	Name of the maintenance window that captured the alert.
custom_info.forward_Alerts	Whether the alert is forwarded to clustering algorithms or not. Set to **`false`** by default.

Maintenance Window Manager Reference

This is a reference for the Maintenance Window Manager Moolet.

Cisco recommends you do not change any properties that are not in this reference guide.

You can change the behavior of the Maintenance Window Manager by editing the configuration properties in the **`$MOOGSOFT_HOME/config/moolets/maintenance_window_manager.conf`** configuration file. It contains the following properties:

name

Name of the Maintenance Window Manager Moolet. Do not change.

Type: String

Required: Yes

Default: **"MaintenanceWindowManager"**

classname

Moolet class name. Do not change.

Type: String

Required: Yes

Default: **"CMaintenance"**

run_on_startup

Determines whether the Maintenance Window Manager runs when Cisco Crosswork Situation Manager starts. By default, it is set to **true**, so that when Moogfarmd starts, it automatically creates an instance of the Maintenance Window Manager.

Type: Boolean

Required: Yes

Default: **true**

metric_path_moolet

Determines whether or not Cisco Crosswork Situation Manager factors the Maintenance Window Manager into the Event Processing metric for Self Monitoring.

Type: Boolean

Required: Yes

Default: **true**

process_output_of

Defines the input source for the Maintenance Window Manager. This determines the Maintenance Window Manager's place in the alert processing workflow.

Type: List

Required: Yes

One of: **AlertBuilder**, **AlertRulesEngine**, **Enricher**

Default: **"AlertBuilder"**

maintenance_status_field

Name of the **custom_info** field or key used to indicate the alert's maintenance status.

Type: String

Required: Yes

Default: **"maintenance_status"**

maintenance_status_label

Value of the **custom_info.maintenance_status** field used to indicate that an alert is in maintenance.

Type: String

Required: Yes

Default: **"In maintenance"**

update_captured_alerts

If enabled, ensures the maintenance status of an alert is set to null once the Maintenance Window that captured it has expired. If disabled, the maintenance status field of a captured alert remains as the text value set in the **maintenance_status_label** property, unless that alert reoccurs when all **custom_info** maintenance fields are set to null.

Type: Boolean

Required: Yes

Default: **true**

It is possible to add a column in the alert view displaying the 'Maintenance Status' for each alert and the text visible in this column can be controlled by editing the maintenance_status_label in the MaintenanceWindowManager Moolet configuration in **$MOOGSOFT_HOME/config/moolets/maintenance_window_manager.conf.**

For the feature to function, you must place the Maintenance Window Manager Moolet before a Sigalising Moolet in a forwarding chain. It is also appropriate for you to locate it before the Alert Rules Engine in the processing chain.

Empty Moolet

The Empty Moolet enables Cisco Crosswork Situation Manager integrators to intercept and handle Message Bus events without impacting upon the existing alert flow logic and processing. This provides a mechanism for you to implement your own alert processing rules. The Empty Moolet can also be used to provide general augmentation of alert and Situation details, for example, Enrichment.

An Empty Moolet can be passed an alert or a Situation by one of the following mechanisms:

- Process output of: The Empty Moolet exists in the alert processing chain.
- Event handler: The Empty Moolet listens for specific message types on the bus.
- Direct forwarding: The Empty Moolet is handed an object by another Moolet, for example, Moolet A forwards an alert to Moolet B.

A single Empty Moolet uses one or more of these mechanisms.

Configure Empty Moolet

The Empty Moolet takes messages off the Message Bus according to message type and passes them to a Moolet. The configuration includes the message types to register interest for and the name of the

Moolet to pass them to. For example, to integrate with an incident management system such as ACMEIncidentManager, the Empty Moolet must:

- Listen to **`NewThreadEntry`** events (the topic on Message Bus is **`/sig/thread/entry/new`**) and **`SigStatus`** events (the topic on Message Bus is **`/sigs/status topic`**).
- Interrogate the events to select only those in which the incident management system has registered an interest via the Graze API **`addSigCorrelationInfo`** request.
- Filter out those events which were originated by the incident management system via the Graze API to avoid sending duplicate information.
- Extract relevant information from the event including the incident management system entity reference.
- Send the information to the incident management system via the REST.V2 Moobot module which supports the sending of simple RESTful POST requests using basic HTTP authentication.

The following example demonstrates an Empty Moolet configuration for this scenario:

```
{
        name                    : "ACMEIncidentManager",
        classname               : "CEmptyMoolet",
        run_on_startup  : true,
        moobot                  : "ACMEIntegration.js",
        event_handlers  : [
                "NewThreadEntry".
                "SigStatus"
        ]
}
```

This example shows one way of integrating Cisco Crosswork Situation Manager with another system. Each integration is dependent upon the individual use cases and systems being integrated.

See Alert Manager for a further example of an Empty Moolet configuration.

Note

Not all event handlers are required for every integration. Only specify required handlers.

Customize Empty Moolet

To invoke custom javascript for a particular set of actions related to Situations, you can leverage the Empty Moolet to listen for these actions and use the data within the Situations involved. For example, when a Situation is closed you may want to notify an external entity via the REST.V2 module.

Edit the configuration file **`moog_farmd.conf`** to associate the CustomTaskRunner Moobot with the Empty Moolet, and listen for SigAction events:

```
{
    name               : "CustomTaskRunner",
    classname          : "CEmptyMoolet",
    run_on_startup     : true,
    metric_path_moolet : false,
    moobot             : "CustomTaskRunner.js",
    event_handlers     : [
        "SigAction"
    ]
}
```

This is an example of Moobot code that runs a function when a supported Situation action occurs in Cisco Crosswork Situation Manager:

CustomTaskRunner.js

```
var events = MooBot.loadModule('Events');
var logger = MooBot.loadModule('Logger');
var constants = MooBot.loadModule('Constants');

logger.debug("Empty Moolet Started.");

/**
 * ### situationAction
 *
 * Listen for specific "sigAction"
 *
 * @param {object} situation - A situation object from Events
 */

function situationAction(situation) {
    logger.warning("Checking Action event...");

    var sitn_id = situation.value("situation_id");
    var action = situation.payload().valueOf("action");

    if (action !== null) {
        var details = situation.getActionDetails();
        // The name of the URL Tool has to match to trigger action
        if (action == "Ran Tool") {
            if (details.tool == urlToolName) {
                runFunction(sitn_id);
            }
        }
    }
}

/**
 * ### runFunction
 *
 * Run some function
 *
 * @param {number} sitn_id - The Situation Id
 */

function runFunction(sitn_id) {
    logger.info('Run some function for Situation Id ' + sitn_id);
}

//
// Listen for SigAction event to see if certain URL tool has been run
//

events.onEvent("situationAction",constants.eventType("SigAction")).listen();
```

The **`urlToolName`** must match the name of the Situation URL tool. The Situation ID is available in the event payload, because the tool is run in the context of a particular Situation.

Enricher Moolet

Enable the Enricher Moolet to update alerts with Enrichment data. See UI Enrichment for further information.UI Enrichment

Configure Enricher

You can define the behavior of the Enricher Moolet by editing the **$MOOGSOFT_HOME/config/moolets/enricher.conf** configuration file.

Enricher Parameters

The parameters that relate to the Enricher Moolet are as follows:

run_on_startup

Determines whether Enricher runs when Cisco Crosswork Situation Manager starts. If enabled, Enricher updates alerts with enrichment data from the moment the system starts, without you having to configure or start it manually.

Type: Boolean

Default: **false**

metric_path_moolet

Determines whether or not Enricher is included in the the Event Processing metric for Self Monitoring.

Type: Boolean

Default: **false**

description

Describes the Moolet.

Type: String

Default: **Alert Enrichment**

The default Enricher parameters are as follows:

```
{
   name               : "Enricher",
   classname          : "com.moogsoft.farmd.moolet.enricher.CEnricherMgr",
   run_on_startup     : true,
   persist_state      : false,
   metric_path_moolet : true,
   process_output_of  : "AlertBuilder",
   description        : "Alert Enrichment"
}
```

Note

nameand**classname**are hardcoded and should not be changed.

Output Parameters

These parameters control the output processed by the Moolet:

process_output_of

Defines the source of the alerts that Enricher processes. By default, the Moolet connects directly to the Alert Builder.

Type: List

One of: `AlertBuilder, AlertRulesEngine` Default: **`AlertBuilder`**

Notifier Moolet

Introduction

The Notifier Moolet enables Cisco Crosswork Situation Manager to act on **`invite`** MooMS Bus messages and optionally send an email.

For example, to send an email when a user is invited to a Situation via the UI, the Notifier Moolet must:

1. Listen to **`Invite`** Events
2. Interrogate the Events to identify Situation invitations
3. Filter out Events for Situations we are not interested in notifying
4. Extract relevant information from the Event including the Situation Id and User Id
5. Send an email message containing a customized body to a recipient using the Mailer Moobot module

Moogfarmd Configuration

The Notifier Moolet is designed to take messages off the MooMS bus according to message type.

You can edit the Notifier in the **`$MOOGSOFT_HOME/config/moolets/notifier.conf`** configuration file.

Moolets are capable of supporting multiple Moobots. By configuring a Moolet to run multiple Moobots, you can customise the functions of the default Moobot. You can use this feature to customise the actions for Workflow Engine. To do this, locate the Moobot property in the Moolet configuration file and add a comma-separated list of the Moobots you want to run. See the extract Notifier Moolet parameters below and notice the default `"moobot"` property which contains one Moobot: `"Notifier.js"`.

The default configuration is as follows:

```
{
    name            : "Notifier",
    classname       : "CNotifier",
    run_on_startup  : false,
    metric_path_moolet : false,
    moobot          : "Notifier.js"
}
```

Teams Manager Moolet

The Teams Manager Moolet is triggered by Cisco Crosswork Situation Manager when a Situation is created, updated and deleted, and also when a team is created and updated. You can assign teams to Situations using the filters in the UI under Settings > Teams > General. If there are no filters for a team, it is assigned all new Situations by default.

If you use the "assignTeamsToSituation" Graze API endpoint or MoogDb method to assign teams to a Situation, Cisco Crosswork Situation Manager marks the Situation as overridden. The Teams Manager Moolet can no longer act on it even if that Situation matches a filter.

You can alter the behavior of the Teams Manager Moolet by changing the "Situation Update Policy" in the UI under Settings > Teams.

One Teams Manager Moolet is run for every instance of Cisco Crosswork Situation Manager.

Configure Teams Manager

You can configure the Teams Manager Moolet in the **$MOOGSOFT_HOME/config/moolets/teams_manager.conf** configuration file.

Moolets are capable of supporting multiple Moobots. By configuring a Moolet to run multiple Moobots, you can customise the functions of the default Moobot. You can use this feature to customise the actions for Workflow Engine. To do this, locate the Moobot property in the Moolet configuration file and add a comma-separated list of the Moobots you want to run. See the extract Teams Manager Moolet parameters below and notice the default "moobot" property which contains one Moobot: "SituationMgr".

Teams Manager Properties

The properties that relate to the Teams Manager Moolet are:

run_on_startup

Determines whether Teams Manager runs when Cisco Crosswork Situation Manager starts. If you enable it, Teams Manager processes Moolet output from the moment the system starts, without you having to configure or start it manually.

Type: Boolean

Default: **true**

metric_path_moolet

Determines whether or not Cisco Crosswork Situation Manager includes Teams Manager in the Event Processing metric for Self Monitoring.

Type: Boolean

Default: **false**

moobot

JavaScript program that controls and customizes the behavior of Teams Manager.

Type: String

Default: **"TeamsMgr.js"**

The default Teams Manager configuration is:

```
name                          : "TeamsMgr",
classname                : "CTeamsMgr",
run_on_startup           : true,
metric_path_moolet       : false,
moobot                   : "TeamsMgr.js",
#
# Specifies the list of all the moolet that can change
```

```
# or create situations. Remove this section if the
# TeamsMgr is running in its own instance.
#
process_output_of       : [
        "Speedbird",
        "Cookbook",
        "Default Cookbook",
        "SituationMgr"
        ]
```

Note

name and **classname** are hardcoded and should not be changed.

Output Parameters

These parameters control the output the Moolet processes:

process_output_of

The Moolets that perform actions that trigger the Teams Manager:

Type: Array

Valid Moolets : Sigaliser, Speedbird, Cookbook, Default Cookbook, SituationMgr

Default: **["Sigaliser", "Speedbird", "Cookbook", "Default Cookbook", "SituationMgr"]**

Scheduler Moolet

You can schedule jobs at regular intervals by editing the **$MOOGSOFT_HOME/config/moolets/scheduler.conf** configuration file:

```
# The Scheduler is used to run scheduled jobs at regular
# intervals throughout the lifetime of moog_farmd. Only this
# moolet, which cannot subscribe to the MooMS bus and
# listen to events, is allowed to submit scheduled jobs.
#
# To start up successfully it must have the name and threads
# values set to "Scheduler" and 1 respectively.
{
        name                    : "Scheduler",
        classname               : "CScheduler",
        run_on_startup          : false,
        metric_path_moolet      : false,
        moobot                  : "Scheduling.js",
        threads                 : 1
}
```

Moolets are capable of supporting multiple Moobots. By configuring a Moolet to run multiple Moobots, you can customise the functions of the default Moobot. You can use this feature to customise the actions for Workflow Engine. To do this, locate the Moobot property in the Moolet configuration file and add a comma-separated list of the Moobots you want to run. See the extract Scheduler parameters above and notice the default "moobot" property which contains one Moobot.

To load the scheduler module:

```
var scheduler   =MooBot.loadModule('Scheduler');
```

Run jobs using, for example:

```
// A job that fails and does not restart.
scheduler.scheduleJob(this, "knockOnce", 5, 5, false);

This calls a method in the same js file called knockOnce:

function knockOnce()
{
    logger.warning("Knock knock");
    throw new Error("Failed to knock.");
}
```

The scheduledJob method has two possible parameter sets:

- scheduleJob(this, functionName, start_delay , interval);
- scheduleJob(this, functionName, start_delay, interval, true | false) ;

Parameter	Description
first	always this
second	the name of the function to call to run the job
third	is the delay from starting farmd to the first run (in seconds)
fourth	the interval between runs (in seconds)
fifth	decides whether the job will run again if it failed previously

Scheduling Frequency

When executing multiple jobs we recommend that you try and offset potential workload, by for example staggering the initial run of multiple jobs a few seconds apart or scheduling jobs at slightly offset frequencies.

Constraints

- Must be single threaded.
- Only one per Moogfarmd process.
- Has to be called Scheduler.
- Use a Moobot module function as a scheduled job - which involves some rarer JavaScript as the scheduler won't recognise function names from modules (it can't find them)

For example, in your scheduler moobot you might have:

```
MooBot.loadModule('AutoClose.js');
var autoClose=new AutoClose();

// Bind the module function locally to the module function.

var autoCloseAlertFunction = autoClose.closeAgedAlerts.bind(autoClose);

// Schedule execution

scheduler.scheduleJob(this, "autoCloseAlertFunction" , 60,
autoCloseAlertFrequency , true);
```

Housekeeper Moolet

The Housekeeper Moolet performs the following tasks:

- Periodic background tasks required for the Auto Close feature and for the moving of data to the historic database. See Configure Historic Data Retention document and the Historic Data Utility Command Reference for more information. Gathers statistics from the system, for example Team Insights.
- The Graph Analyser task to calculate the Vertex Entropy for the nodes in your topologies. See Create and Manage Topologies and Set Up Vertex Entropy for further information. To verify that the Housekeeper Moolet is running, use the HA Control utility with the view argument:

```
ha_cntl -v
```

See HA Utility Command Reference for more information about this utility.

Configure the Housekeeper Moolet

You can define the behavior of the Housekeeper Moolet by editing the configuration file: **`$MOOGSOFT_HOME/config/moolets/housekeeper.conf`**

Moolets are capable of supporting multiple Moobots. By configuring a Moolet to run multiple Moobots, you can customise the functions of the default Moobot. You can use this feature to customise the actions for Workflow Engine. To do this, locate the **`moobot`** property in the Moolet configuration file and add a comma-separated list of the Moobots you want to run.

Housekeeper properties

The Housekeeper Moolet properties are as follows.

run_on_startup

Determines whether Housekeeper runs when Cisco Crosswork Situation Manager starts. If set to true, Housekeeper performs its background tasks from the moment the system starts, without you having to configure or start it manually.

Type: Boolean

Default: **`true`**

metric_path_moolet

Determines whether Housekeeper is factored into the event processing metric for Self Monitoring. See Moogfarmd Reference for more information.

Type: Boolean

Default: **`false`**

standalone_moolet

Determines whether the Housekeeper can listen for events generated by other Moolets within the same Moogfarmd instance without being in a processing chain.

Type: Boolean

Default: **`true`**

moobot

A comma-separated list of Moobots the Housekeeper Moolet will run.

Type: String

Default: Housekeeper.js

Example configuration

An example Housekeeper Moolet configuration as follows:

```
{
    run_on_startup     : true,
    metric_path_moolet : false,
    standalone_moolet  : true,
    moobot             : "Housekeeper.js"
}
```

Situation Manager

The Situation Manager listens for Situation creation, update, or closure actions and passes the Situation to an associated Moobot. It runs as a standalone Moolet by default.

You can define the algorithms for which the Situation Manager processes output, the Moobot it passes Situations to and the actions performed on those Situations.

When a Moobot receives a Situation, you can configure it to perform functions such as data enrichment, auto-assignment to a user or notifying a third-party tool to raise a ticket.

Configure the Situation Manager

The Situation Manager configuration file is located here: **`$MOOGSOFT_HOME/config/moolets/situation_manager.conf`**. You can define the following properties:

run_on_startup

Determines whether Situation Manager starts when Cisco Crosswork Situation Manager starts.

Type	String
Required	Yes
Default	N/A

metric_path_moolet

Determines whether Situation Manager is included in the events processing metric for Self Monitoring.

Type	Boolean
Required	No
Default	False

moobot

Determines which Moobot receives Situations from the Situation Manager. The Moobot JavaScript files are located here: **`$MOOGSOFT_HOME/bots/moobots`**.

Moolets are capable of supporting multiple Moobots. By configuring a Moolet to run multiple Moobots, you can customise the functions of the default Moobot. You can use this feature to customise the actions for Workflow Engine. To do this, locate the Moobot property in the Moolet configuration file and add a comma-separated list of the Moobots you want to run. See the extract Situation Manager parameters below and notice the default "moobot" property which contains one Moobot: "SituationMgrLabeller.js".

Type	String
Required	Yes
Default	SituationMgr.js

standalone_moolet

Determines whether Situation Manager runs as a standalone Moolet.

Type	Boolean
Required	No
Default	True

Example

An example Situation Manager Moolet configuration as as follows:

```
{
    name               : "SituationMgr",
    classname          : "CSituationMgr",
    run_on_startup     : true,
    metric_path_moolet : false,
    moobot             : "SituationMgrLabeller.js",
    standalone_moolet  : true
}
```

Note

Do not change the name and classname properties.

Configure the Moobot

Situation Manager listens for three event types by default:

- Sig: Situation creation.
- SigClose: Situation closure.
- SigUpdate: Situation update.

If you want to listen for other events, create an Empty Moolet and define the events in the **event_handler**. See the **eventType** method in Constants for a full list of event types.

You can listen for specific Situation actions using the SigAction event. It can filter our the following actions:

Assigned Moderator	**Deacknowledged Situation**	**Described Situation**
Situation Resolved	**Moderator**	**Excluded User**
Situation Revived	**Added Alerts To Situation**	**Invited User**

Situation Closed **Assigned Queue** **Created By Merge** **Used In Merge** **Created By Split** **Used For Split** **Ran Tool** **Acknowledged Situation** **Moderator**	**Added Entry To Thread** **Changed Situation Processes** **Changed Situation Services** **Created Thread** **Agreed With Thread Entry** **Commented On Thread Entry** **Disagreed With Thread Entry** **Changed Situation Custom Info**	**Moved Alerts To Situation** **Removed Alerts From Situation** **Situation Teams Changes** **Marked Thread Entry As Resolving** **Unmarked Thread Entry As Resolving** **Situation Rating** **Situation Rating Removed**

The Situation Manager can send Situations to one of three Moobots. You can customize these to meet your requirements:

- Situation Manager: Situation Manager's default associated Moobot. The configuration file is **`SituationManager.js`**. For information on how to configure it, see Moobot Modules.
- Situation Manager Labeler: You can use the Situation Manager Labeler to enrich Situations by dynamically adding alert properties to the Situation description. The configuration file is **`SituationMgrLabeller.js`**. See Situation Manager Labeler for more information.
- Situation Manager Netcool: This Moobot is required for the Netcool legacy LAM. The configuration file is **`SituationMgrNetcool.js`**. For more information see Netcool Legacy LAM.

Services

In Cisco Crosswork Situation Manager a service represents a supportable unit that provides a set of related functionality. A service may relate to a single application or it may incorporate multiple applications. Example services may include web application, web service, data management, database, network.

This document outlines how to create services, assign them to Situations, associate them with teams and monitor affected services in the Cisco Crosswork Situation Manager UI.

Before You Begin

Before you begin to create services in Cisco Crosswork Situation Manager, ensure you have met the following requirements:

1. Identify the services in your environment. A third party tool or external system may be useful for this task, for example the list of business services, applications or assignment groups in ServiceNow.
2. If your service data is held externally to Cisco Crosswork Situation Manager, identify the data source.
3. Choose one or more methods that you will use to create and assign services:
    - Graze API: Useful when you have a known list of services that change infrequently.
    - Situation Manager Labeller: Useful when your services are likely to change and you want to avoid the overhead associated with manual creation and assignment.
    - Moobot: Useful when you are already using a custom Moobot for enrichment. See Enrichment for more information.
    - Another Enrichment method: Another method may be suitable depending on the source of your service data, for example a static data file. See Enrichment for more information.
    - Cisco Crosswork Situation Manager UI: An administrator can assign services to individual Situations in the UI.

Create Services and Assign Services to Situations

You can use one of the following methods, or a combination of these, to add services and assign them to Situations.

Graze API Endpoints

The addService endpoint enables you to create a single service or script the creation of multiple services. You can use setSituationServices to add one or more services to a Situation and getSituationServices to return a list of impacted services for a specified Situation.

See Graze API for details on the command syntax and examples.

Situation Manager Labeller

This utility allows you to create services from your custom data as it is ingested into Cisco Crosswork Situation Manager and assign those services to Situations.

See Create Services With Situation Manager Labeller for more information and an example.

Moobot

If you are using a custom Moobot to enrich on Situation creation, you can use the MoogDb addService and setSituationServices methods to create services as part of this process. See Enrichment for further information.

Another Enrichment Method

See Enrichment for further information on other enrichment methods.

Cisco Crosswork Situation Manager UI

In the UI, go into a Situation Room. Click Services Impacted at the top of the screen to add or remove services from the Situation. You will need administrator rights to perform this function.

Assign Services to Teams

Cisco Crosswork Situation Manager can automatically assign Situations to teams based upon the service data. You can also automatically create teams based on the service data in Situations.

See Manage Teams for details.Manage Teams

Monitor Affected Services

The Services Overview in the UI Workbench Summary allows you to view the impacted services with the highest severity Situations. You can use this information to prioritize which Situations to address first.

See Check Impacted Services for details.Check Impacted Services

Workflow Engine Moolets

The Workflow Engine Moolets perform tasks on events, alerts, and Situations as specified in a user-defined workflow. See Workflow Engine for more information.Workflow Engine

The following files define the actions that are available when you define a workflow in the Cisco Crosswork Situation Manager UI:

- A Workflow Engine Moobot specifies a set of actions that are available when you define a workflow in the Cisco Crosswork Situation Manager UI.
- The **$MOOGSOFT_HOME/config/moolets/** folder has one default config file for each workflow engine:
    - **event_workflows.conf**: Event workflows process event data after data ingestion from a LAM and before the Alert Builder.
    - **enrichment_workflows.conf**: Enrichment workflows process alert data after the Alert Builder but before the Maintenance Window Manager.
    - **alert_workflows.conf**: Alert workflows process alert data after the Maintenance Window Manager and before they pass to a clustering algorithm.
    - **alert_inform_workflows.conf**: Alert Inform workflows trigger specific workflows from within the Cisco Crosswork Situation Manager UI in response to another action on an alert.
    - **situation_workflows.conf**: Situation workflows process Situation data after the Teams Manager. For example, you can use a Situation workflow when you want to integrate with a ticketing system.
    - **situation_inform_workflows.conf**: Situation Inform workflows trigger specific workflows from within the Cisco Crosswork Situation Manager UI in response to another action on a Situation.
    - If you have installed the Add-ons, you will see additional workflow engine configurations.
    - Each configuration file has a **moobot** field that specifies the set of supported Moobots. The default Moobot for all four Moolet types is **$MOOGSOFT_HOME/bots/moobots/WorkflowEngine.js**. Do NOT modify the Cisco supplied WorkflowEngine.js.

You can add and update Workflow Engine functionality. See Install Add-ons for more information.

If you want to create your own Workflow Engine Moolet, see Create a Workflow Engine Moolet.

Create a Workflow Engine Moolet

Implementers and administrators can use the Workflow Engine to add custom logic for event, alert, and Situation processing in Cisco Crosswork Situation Manager.

A Moolet is an intelligence module that performs specific services in Cisco Crosswork Situation Manager.

You can define a workflow in a Workflow Engine (WFE) Moolet to perform tasks on events, alerts, and Situations.

If the existing WFE Moolets do not perform the service that you need, you can create a new WFE Moolet.

Before you create a new WFE Moolet, consider where the new Moolet:

- Fits into your workflow.
- Receives data from and sends data to, and what the data types are.

WFE Moolet types

There are three types of WFE Moolet:

- Event WFE Moolets process event data after that data is ingested by an integration or LAM and before it is processed by the Alert Builder.
- Alert WFE Moolets process alert data after the Maintenance Window Manager and before that data passes to a clustering algorithm.
- Situation WFE Moolets process Situation data after the Teams Manager finishes processing those Situations.

You create a new Moolet by creating a Moolet configuration file in **$MOOGSOFT_HOME/config/moolets/**.

WFE Moolet configuration file examples

For a full description of WFE Moolet configuration file properties, see the WFE Moolet Reference.

Event WFE Moolet

```
name               : "CustomEventMoolet",
classname          : "com.moogsoft.farmd.moolet.workflowengine.CWorkflowEngine",
run_on_startup:    : true,
metric_path_moolet : true,
moobot             : "WorkflowEngine.js",
event_handlers     : "event",
message_type       : "event"
```

Alert WFE Moolet

```
name               : "CustomAlertMoolet",
classname          : "com.moogsoft.farmd.moolet.workflowengine.CWorkflowEngine",
run_on_startup:    : true,
metric_path_moolet : true,
moobot             : "WorkflowEngine.js",
process_output_of  : "MaintenanceWindowManager",
message_type       : "alert"
```

Situation WFE Moolet

```
name               : "CustomSituationMoolet",
classname          : "com.moogsoft.farmd.moolet.workflowengine.CWorkflowEngine",
run_on_startup:    : true,
metric_path_moolet : true,
moobot             : "WorkflowEngine.js",
process_output_of  : "SituationMgr",
message_type       : "situation"
```

Activate the new Moolet

Once you have created the Moolet configuration file:

- Add the new Moolet to the Moogfarmd configuration file at **$MOOGSOFT_HOME/config/moog_farmd.conf**. For example:

```
moolets : [
      ...
      {
        include : "alert_builder.conf"
      },
```

```
    ...
    {
      include : "teams_manager.conf"
    },
    {
      include : "NEW_WFE_MOOLET_NAME.conf"
    }
  ]
```

- Restart the Moogfarmd instance to activate the new Moolet. See Control Processes for more information.Control Processes

Workflow Engine Moolet reference

This is a reference for Workflow Engine (WFE) Moolets. The WFE Moolet configuration files are located at **`$MOOGSOFT_HOME/config/moolets`**. See Create a WFE Moolet for examples of these configuration files.

name

The name of the new Moolet. You must set a unique name for the Moolet name within each Moogfarmd instance.

Type	String
Required	Yes
Default	N/A

classname

The name of the java class that implements the moolet. WFE moolets are always the "com.moogsoft.farmd.moolet.workflowengine.CWorkflowEngine" java class.

Type	String
Required	Yes
Default	N/A

run_on_startup

Determines whether the new Moolet starts when the Moogfarmd instance starts.

Type	Boolean
Required	No
Default	**`true`**

metric_path_moolet

Determines whether the Moolet is included in the events processing metric for Self Monitoring.Self Monitoring

You can set the property to **`true`** or **`false`**. If the new Moolet is in a processing chain and logically before the Alert WFE in that chain, set this property to **`true`**. Otherwise, set it to **`false`**.

Type	Boolean
Required	No
Default	**`false`**

moobot

A Moobot name or list of Moobot names that the new Moolet runs. The Moolet loads the Moobots in the order specified.

Type	Single Moobot: String Multiple Moobots: Array
Required	Yes
Default	N/A

event_handlers

Specifies the type of event and message the new Moolet listens for.

Type	Single event and message: String Multiple events and messages: Array
Required	No
Default	N/A

process_output_of (alert and Situation WFE Moolets only)

A Moolet name or list of Moolet names for the new Moolet to receive events from.

Type	Single Moolet: String Multiple Moolets: Array
Required	No
Default	N/A

message_type

Specifies the type of object that the Moolet processes. There are three options:

- Event: "event".
- Alert: "alert".
- Situation: "situation".

Type	String
Required	Yes
Default	N/A

Alert Manager

The Alert Manager uses the Empty Moolet to enable Cisco Crosswork Situation Manager administrators or implementers to incorporate additional alert processing not handled by the Alert Builder, Maintenance Window Manager or Alert Rules Engine. You can use the Alert Manager in standalone mode or as part of the alert processing workflow.

Configure the Alert Manager

You can edit the Alert Manager configuration file at **$MOOGSOFT_HOME/config/moolets/alert_manager.conf**

Alert Manager properties

The configurable Alert Manager properties are as follows. Do not edit the **name** and **classname** properties in the file.

run_on_startup

Determines whether Alert Manager runs when Cisco Crosswork Situation Manager starts. If set to true, the Alert Manager performs its background tasks from the moment the system starts, without you having to configure or start it manually.

Type	Boolean
Default	**true**

metric_path_moolet

Determines whether Alert Manager is factored into the event processing metric for Self Monitoring. See Moogfarmd Reference for more information.Self Monitoring

Type	Boolean
Default	**false**

moobot

A single Moobot or comma-separated list of Moobots the Alert Manager will run. Specify a JavaScript file or files located in **$MOOGSOFT_HOME/moobots**.

Type	String
Default	**AlertMgr.js**

standalone_moolet

Determines whether the Alert Manager is run as a standalone process or whether it is a component within the alert processing workflow.

Type	Boolean
Default	**true**

event_handlers

Configure the Alert Manager to receive the specified event types. See Constants for more information on event types.

Type	String
Default	**["AlertClose", "AlertUpdate", "Alert"]**

Example configuration

The default configuration file contains an example implementation of the Empty Moolet functionality in the form of the Alert Manager Moolet. For example:

```
{
   run_on_startup     : false,
   metric_path_moolet : false,
   standalone_moolet  : true,
   moobot             : "AlertMgr.js",
   event_handlers     : [ "AlertClose", "AlertUpdate", "Alert" ]
}
```

Alert Manager Moobot

Cisco Crosswork Situation Manager provides a Moobot for the Alert Manager Moolet named **`AlertMgr.js`**. You can use this Moobot to enable a specific action on different alert types. For example, to update a Situation's services when an alert that contains certain attributes is updated.

Empty Moolet

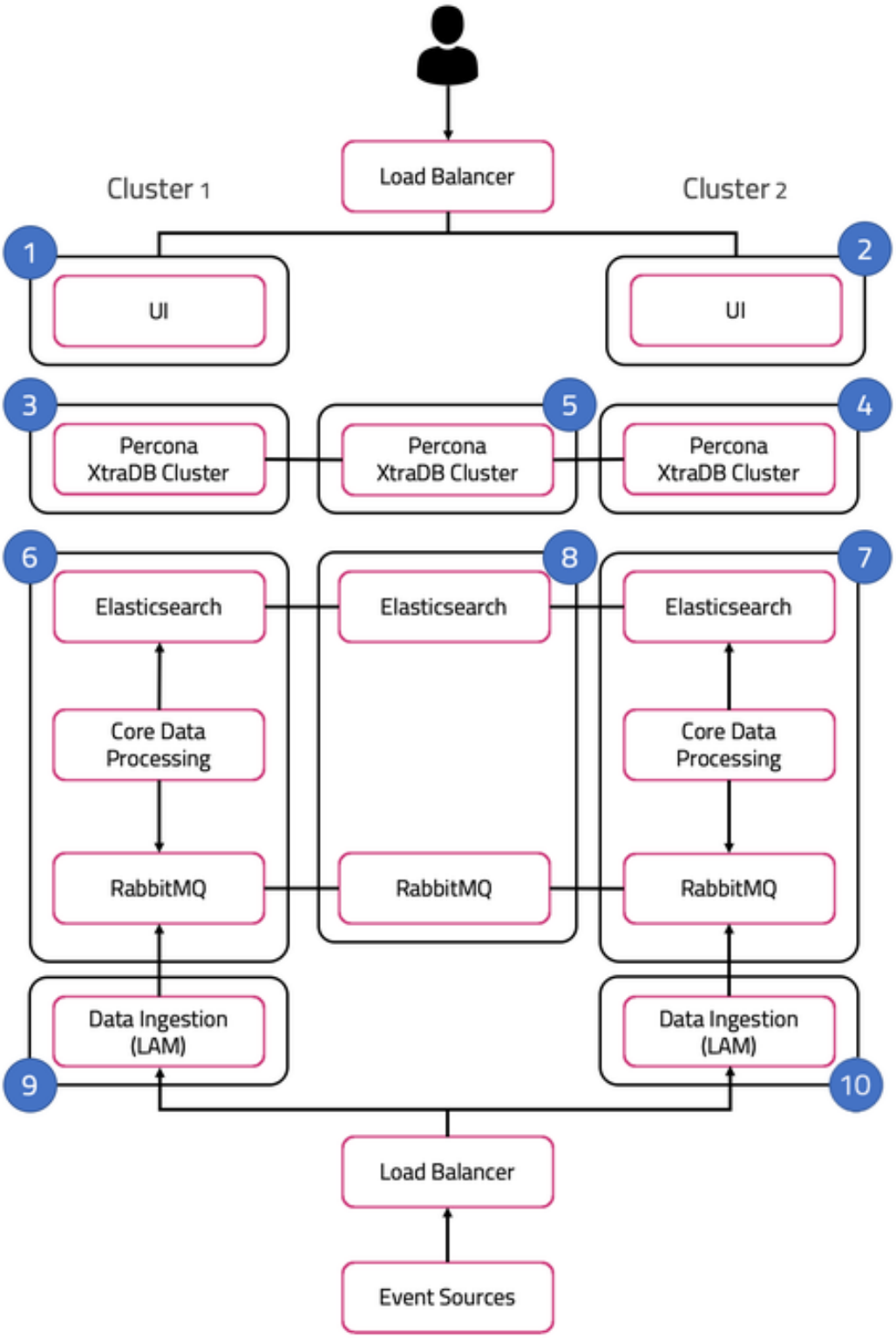
For further information on customizing Cisco Crosswork Situation Manager using the Empty Moolet, see Empty Moolet.

Server Roles

In order to plan your Cisco Crosswork Situation Manager deployment, it helps to understand the different components of Cisco Crosswork Situation Manager and the options for distributing them among multiple physical or virtual machines.

A server role within an Cisco Crosswork Situation Manager installation is a functional entity containing components that must be installed on the same machine. You can distribute different roles to different machines.

The following diagram illustrates the typical deployment strategy for the components of Cisco Crosswork Situation Manager in an High Availability Overview configuration:


Load Balancer
Cluster 1
Cluster 2
1
UI
2
UI
3
Percona XtraDB Cluster
5
Percona XtraDB Cluster
4
Percona XtraDB Cluster
6
Elasticsearch
8
Elasticsearch
7
Elasticsearch
Core Data Processing
Core Data Processing
RabbitMQ
RabbitMQ
RabbitMQ
Data Ingestion (LAM)
Data Ingestion (LAM)
9
10
Load Balancer
Event Sources

The architecture is built upon two clusters with software components that serve several roles. See also HA Reference Architecture.

In the case of a single-server installation, you install all the roles on one machine.

UI role

The UI role comprises Nginx and Apache Tomcat, represented in the diagram as numbers 1 and 2. The Cisco Crosswork Situation Manager servlets groups run in active / active configuration.

Ngnix is the proxy for the web application server and for integrations.

Tomcat is the web application server. It reads and writes to the Message Bus and the database.

Database role

Percona XtraDB Cluster serves the database role, represented in the diagram as numbers 3, 4, and 5. The cluster runs in active / active standby / active standby mode.

Percona Xtra Db Cluster is the system datastore that handles transactional data from other parts of the system: LAMs (integrations), data processing, and the web application server.

HA Proxy handles database query routing and load balancing.

Core role

The Core role, represented by numbers 6 and 7 in the diagram comprises the following:

- Moogfarmd, the Cisco Crosswork Situation Manager data processing component. Moogfarmd consumes messages from the Message Bus. It processes event data in a series of servlet-like modules called Moolets.
- Moogfarmd reads and writes to the database and publishes messages to the bus.
- RabbitMQ which provides the message queue. It receives published messages from integrations. It publishes messages destined for data processing (Moogfarmd) and the web application server.
- Elasticsearch which provides the UI search capability. It indexes documents from the indexer Moolet in the data processing series. It returns search results to Tomcat.

In HA deployments, Moogfarmd automatically runs in active / passive mode. See High Availability Overview for more information.

In concert with the the Redundancy Role server, RabbitMQ and Elasticsearch run in active / active / active mode.

Redundancy role

The redundancy role, represented by number 8 in the diagram, provides the third node required for true HA for RabbitMQ and Elasticsearch.

Data ingestion role

Link Access Modules (LAMs) make up the data ingestion role represented by numbers 9 and 10 in the diagram. Receiving LAMs listen for events from monitoring sources and Polling LAMs poll monitoring sources for events. Both parse and encode raw events into discrete events, and then write the discrete events to the Message Bus.

In HA deployments, receiving LAMs run in active / active mode, but polling LAMs run in active / passive mode.

Load balancers

The load balancers in front of the UI server role and the data ingestion server role are the customer's responsibility.

Severity Reference

Severity is a measure of the seriousness of an event and indicates how urgently it requires corrective action.

Cisco Crosswork Situation Manager LAMs and integrations use six industry standard severity levels as follows:

- 0: Clear - One or more events have been reported but then subsequently cleared, either manually or automatically.
- 1: Indeterminate - The severity level could not be determined.
- 2: Warning - A number of faults with the potential to affect services have been detected.
- 3: Minor - A fault that is not affecting services has been detected. Action may be required to prevent it from becoming a more serious issue.
- 4: Major - A fault is affecting services and corrective action is required urgently.
- 5: Critical - A serious fault is affecting services and corrective action is required immediately.

The severity mapping is set in each LAM configuration file:

```
severity:
{
    "CLEAR"         : 0,
    "INDETERMINATE" : 1,
    "WARNING"       : 2,
    "MINOR"         : 3,
    "MAJOR"         : 4,
    "CRITICAL"      : 5,
}
```

The LAM takes the severity string in a received event and translates it into one of the above integer values using the mapping in its configuration file:

```
sevConverter:
{
    lookup  : "severity",
    input   : "STRING",
    output  : "INTEGER"
},
mapping:
    rules:
```

```
[
    {
        name: "severity",
        rule: "$severity",
        conversion:"sevConverter"
    }
]
```

You can customize the severity section of the LAM configuration file according to the severities used in the system sending events to Cisco Crosswork Situation Manager. In the following example, events sent to the LAM with non-standard severities 'info' and 'Information' are mapped to 'INDETERMINATE' in Cisco Crosswork Situation Manager:

```
severity:
{
    "info"              : 1,
    "Information"       : 1,
    "user"              : 1,
    "warning"           : 2,
    "Warning"           : 2,
    "error"             : 5,
    moog_lookup_default  : 1
}
```

The **`moog_lookup_default`** property specifies a default value to use when the severity does not match any of the defined strings. If you do not set a default, events with an unmapped severity are not processed. For more information on mapping see "Conversion Rules" in Data Parsing.Data Parsing

Cisco Crosswork Situation Manager determines a Situation's severity from the member alert with the highest severity level.

Status ID Reference

The status of alerts and Situations is determined by their status ID. These statuses are used within the Heartbeat Monitor.

The different **`status_id`** values are as follows:

Status ID	Name
1	Opened
2	Unassigned
3	Assigned
4	Acknowledged
5	Unacknowledged
6	Active
7	Dormant
8	Resolved

9	Closed
10	SLA Exceeded

Situation Manager Labeler

You can use the Situation Manager Labeler to set Situation descriptions and fields dynamically, based on the alert data in each Situation. For example, suppose you are defining a correlation based on the **custom_info.services** alert field. To generate descriptions for the resulting Situations, you can specify a label string in the description field such as:

```
$$COUNT(custom_info.services) services affected including
$$CITED(custom_info.services,3)
```

Given this string, the resulting descriptions include the three most-cited services and the number of times each service is cited by a member alert:

```
5 services affected including cust-login(7), verify-login(6), update-login-
info(4), ...
```

Note

The Situation Manager Labeler is installed by default with Cisco Crosswork Situation Manager v7.3 and higher. For previous releases, contact Cisco Customer Support to obtain installers and instructions.

Usage

Given a macro operation and an alert data field, the operation iterates through the relevant values in the Situation alerts and returns a string derived from these values.

The usage for fields with single values (prefix is one $): **$macro(alert-field, max-alerts-to-include)**

The usage for fields with lists (prefix is two $'s): **$$macro(alert-field, max-alerts-to-include)**

The **max-alerts-to-include** field is optional. This value limits the number of alert values to include in the description.

Consider the following example. You want to create a label with a count of all the affected services (**custom_info.services**) cited in all alerts. A Situation has two alerts:

- Alert 1: **custom_info.services = [a, b, c];**
- Alert 2: **custom_info.services = [d, e, f];**

$COUNT treats the fields as individual values and returns a count of 2.

$$COUNT treats the fields as lists of individual values and returns a count of 6.

Update Situation descriptions

You can use the following macros to generate Situation descriptions. These macros are supported for single values (**$macro**) and lists (**$$macro**):

- `COUNT(alert-field)` -– Return the count of alert-field citations, including duplicates.
- `UCOUNT(alert-field)` -– Return the count of unique alert-field citations, excluding duplicates.

- `CRITICAL(alert-field)` -– Return the string **`CRITICAL :`** if any alerts have a severity of critical, or 5. This macro is only useful for the **`severity`** field.
- `UNIQ(alert-field)` -– Return a list of all cited **`alert-field`** values.
- `TOP(alert-field)` -– Return the **`alert-field`** value cited by the most alerts in the Situation.
- `CITED(alert-field)` -– Return a list of the unique **`alert-field`** values cited by alerts in the Situation along with the number of times they are cited `--` for example, **`source1 (10)`**, **`source5 (7)`**, **`source3 (4)`**.
- `CITEDLIST(alert-field)` -– Same as **`$CITED`** but returns a string instead of a JSON list.
- `BOOLEAN(alert-field)` -– Return false if all values are "falsy:" 0, null, undefined, `""`, and so on.
- `$CLASS(custom-info-value)` -–Set the situation **`custom_info.situationClass`** field with this value and include this value in the situation label. This is useful for specifying custom information about the Situation such information about the clustering algorithm -– for example, **`cookbook-name.recipe-name`**.
- `TOLIST(alert-field)` -– Creates a comma-separated string from the elements of **`alert-field`**.

Note

UI list-based filtering is now native, so **`$TOLIST()`** should no longer be required.

Numeric fields only

The following macros are supported for numeric fields only, such as **`time`**, **`severity`**, or **`event-count`**.

- `MIN(alert-field)` -– Return the minimum cited value of **`alert-field`**.
- `MAX(alert-field)` -– Return the maximum cited value of **`alert-field`**.
- `AVE(alert-field)` -– Return the average of all cited values of a**`alert-field`**.
- `SUM(alert-field)` -– Return the average of all cited values of **`alert-field`**.
- `NUM(alert-field)` -– Return the set of **`alert-field`** values sorted numerically from low to high, including duplicates.
- `UNUM(alert-field)` -– Return the set of unique **`alert-field`** values sorted numerically from low to high, excluding duplicates.

Text fields only

The following macros are supported for text fields only, such as **`service`**, **`source`**, or **`description`**.

- `ALPHA(alert-field)` -– Return the set of alert-field values sorted alphabetically, including duplicates.
- `UALPHA(alert-field)` -– Return the set of unique alert-field values sorted alphabetically, excluding duplicates.

List values only

The following macros are supported for array values only.

- `$$INTERSECT(alert-field)`-– Return the list of intersections -- that is, alert-field values cited by multiple alerts. This macro parses the alert-field array values and returns a list of the items with multiple citations.For example, support a Situation has two alerts. The service field of alert 1 is **`[a, b, c]`**. The service field of alert 2 is **`[b, c, d]`**. **`$$INTERSECT(service)`** would return the list **`[b, c]`**.
- `$$NINTERSECT(alert-field)`-– Return the number of intersections. Given the previous example, **`$$NINTERSECT(service)`** would return the number 2.
- `$$CINTERSECT(alert-field)` -– Return the list of common intersections -- that is, values cited by all alerts in the Situation. This macro is useful for identifying a possible root cause that caused all the alerts to get correlated together.

Limiting the number of alerts to consider

By default, each macro considers all alerts in a Situation up to a maximum of 200. You might want to specify a lower threshold to ensure that labeling does not become a bottleneck in systems with large or frequently-updated Situations. To lower the threshold, append the **`$FETCH`** modifier at the start of the Labeler string:

```
$FETCH(max-alerts-to-consider)Labeler-string
```

For example, the following macro considers the first alert in each Situation based on alert ID:

```
$FETCH(1) Application Situation for: $UNIQ(custom_info.application) at
DataCentre $UNIQ(custom_info.location)
```

You should specify the maximum number of alerts needed to ensure an accurate description. If you are correlating based on a specific field such that all alerts have the same value for that field, you only need to fetch 1 alert.

Warning

Do not specify a fetch value higher than 20.

Update Situation columns

You can use the following macros to update columns in the Situation Table with values contained in its member alerts.

- `$$SERVICES(alert-field)` -–Update the Services Impacted column in the Situation with all unique alert-field values cited in the member alerts.
- `$$ISERVICES(alert-field)` -– Update the Services Impacted column in the Situation with all unique alert-field values cited in 2 or more member alerts.
- `$$PROCESSES(alert-field)`-– Update the Processes Impacted column in the Situation with all unique alert-field values cited in the member alerts.

You can also use the **`$MAP[ ]`** macro to update a **`custom_info`** field in the Situation with data from the member alerts. The usage is as follows:

```
$MAP[ $MACRO(source alert field, destination custom_info field) ]
```

Update Situation fields

You can use the following macro to update the **custom_info** field for individual Situations.

- `$MAP[source-alert-field, destination-custom-info-field]` -–Update a **custom_info** field in the Situation with data from the member alerts. You can include multiple macros in the same MAP macro, as shown in the following example:
- **$MAP[$UNIQ(source, hosts) $UCOUNT(source, num_hosts)]**
- `$CLASS(custom-info-value)` -–Set the situation **custom_info.situationClass** field with this value. This is useful for specifying custom information about the situation, such as information about the clustering algorithm -– for example, **cookbook-name.recipe-name**.

Example

For instructions on how to use the Situation Manager Labeler to automatically create services based on **custom_info** data, see Create Services With Situation Manager Labeller

Workflow Engine

Implementers and administrators can use a workflow engine to add custom logic for event, alert, and Situation processing in Cisco Crosswork Situation Manager. You can check conditions against object data, modify object data, and control object routing. For example you can set up a workflow engine to process and normalize data from a LAM or Integration before it forwards the data to the Alert Builder.

Some scenarios where you can implement the Workflow Engine include:

- Using the Alert Workflow Engine to escalate alerts. For example to respond to a disk space alert.
- Using the Event Workflow Engine to prevent them from processing.

See for an example.

- Enrich alerts with external data such as the services supported by a host or its location.
- Using the Heartbeat Workflow Engine to detect the absence of events like a missing keep alive event from a predictable source.
- See for an example.
- Controlling stateful workloads. For example, you can configure a workflow to hold a "link down" event until Cisco Crosswork Situation Manager receives a corresponding "link up" event within a time limit.
- Using the Situation Workflow Engine to integrate with external systems for ticketing, notification, automation, and reporting.

See for an example.

Default Workflow Engine types

Each workflow engine is a Moolet that operates within the context of Moogfarmd. Cisco Crosswork Situation Manager includes the following workflow engines by default:

- Event Workflows: Process events data after data after ingestion from a LAM and before the Alert Builder.
- Enrichment Workflows: Process alerts after the Alert Builder but before the Maintenance Window Manager.
- Alert Workflows: Process alerts after the Maintenance Window Manager and before they pass to clustering algorithms. For example, you can use an alert workflow when you want to route a specific type of alert to a specific clustering algorithm.
- Alert Inform Workflows: Invokes a specific alert workflow outside of the Alert Workflow Engine chain.
- Situation Workflows: Process Situations after the Teams Manager. For example, you can use a Situation workflow when you want to integrate with a ticketing system.
- Situation Inform Workflows: Invokes a specific Situation workflow outside of the Situation Workflow Engine chain.

Cisco periodically releases new workflow engines. See Add-ons for the most recent update.

Because workflow engines are Moolets, you can use a Moobot to extend the engine functionality. You can create a new Workflow Engine Moolet if the existing Moolets do not perform the service that you need. You can also add new workflow engines. For more information, see Workflow Engine Moolets.

Workflows

Each workflow engine is a container for a set of workflows. In general a workflow comprises the following:

- A workflow definition that defines the workflow purpose, the objects it affects, and some basic functionality.
- Workflow actions that let you apply functions to object data and apply some routing rules in case the function returns false. There are functions to let you check conditions of object data within the workflow, modify the object data, and route the object within the workflow or to another Moolet.

For most engines, workflows execute in numerical order from first to last. You can rearrange the order of the workflows to control the order of execution within the engine. The exception are the "inform" workflows. Inform engines let you execute a single, specific workflow without executing all the workflows in the engine.

Within a workflow, actions execute in numerical order.

Each action within a workflow lets you control the flow in the case that its function returns false.

Cisco Crosswork Situation Manager ships with the following default workflows:

- Closed Objects Filters to prevent processing of closed ob in the Alert Workflows and Situation Workflows.
- Automated Ticketing to help you integrate with third party ticketing systems in the Situation Workflows.

To learn how to create workflows, see Manage Workflows. For information on creating actions within a workflow, see Manage Workflow Engine Actions.

Manage Workflow Engine Actions

After you have defined the data you want to process using a Workflow Engine in Cisco Crosswork Situation Manager, you can set up actions to programmatically transform the data and control the data flow.

Read through Workflow Engine Strategies and Tips for ideas about how to use the Workflow Engine.

When you edit an engine, you can click +Add Action to create a new action, or double-click an existing action to edit it.

Delay

By default, each workflow has a delay action. It is mandatory and you can not delete it. You can set the delay for up to 86,400 seconds (24 hours). If you enable the Reset option, if another matching event reaches the delay, the delay timer resets to 0.

Actions

Define workflow actions as follows to add custom processing to events, alerts, or Situations depending on the type of engine:

Action Property	Description
Action Name	Identifier for the action. Must be unique within the workflow.
Function	A programmatic task based on JavaScript and Java functions. The available function list varies according to the object: event, alert, enrichment, or Situation. When you set the function, the UI displays its description and updates the Values to correspond to the function. For example, the **`contains`** action lets you check a **`field`** in your object for matching **`values`**. See Workflow Engine Functions Reference.Workflow Engine Functions Reference
Value	Parameters for the function. The parameters vary from function to function. When you select a function, the UI updates the description of the parameters. For more information, see Workflow Engine Functions Reference and Alert and Event Field Reference.Workflow Engine Functions Reference
Forwarding Behavior	Controls the data flow. For objects where the function returns **`false`**, you can choose to always forward to the next action or workflow, stop the current workflow, or stop all workflows for the object.

If you have multiple actions, you can drag and drop them to arrange them according to your requirements.

Workflow Engine Moolets

The Workflow Engine Moolets perform tasks on events, alerts, and Situations as specified in a user-defined workflow. See Workflow Engine for more information.Workflow Engine

The following files define the actions that are available when you define a workflow in the Cisco Crosswork Situation Manager UI:

- A Workflow Engine Moobot specifies a set of actions that are available when you define a workflow in the Cisco Crosswork Situation Manager UI.

- The **$MOOGSOFT_HOME/config/moolets/** folder has one default config file for each workflow engine:
    - **event_workflows.conf**: Event workflows process event data after data ingestion from a LAM and before the Alert Builder.
    - **enrichment_workflows.conf**: Enrichment workflows process alert data after the Alert Builder but before the Maintenance Window Manager.
    - **alert_workflows.conf**: Alert workflows process alert data after the Maintenance Window Manager and before they pass to a clustering algorithm.
    - **alert_inform_workflows.conf**: Alert Inform workflows trigger specific workflows from within the Cisco Crosswork Situation Manager UI in response to another action on an alert.
    - **situation_workflows.conf**: Situation workflows process Situation data after the Teams Manager. For example, you can use a Situation workflow when you want to integrate with a ticketing system.
    - **situation_inform_workflows.conf**: Situation Inform workflows trigger specific workflows from within the Cisco Crosswork Situation Manager UI in response to another action on a Situation.
    - If you have installed the Add-ons, you will see additional workflow engine configurations.
    - Each configuration file has a **moobot** field that specifies the set of supported Moobots. The default Moobot for all four Moolet types is **$MOOGSOFT_HOME/bots/moobots/WorkflowEngine.js**. Do NOT modify the Cisco supplied WorkflowEngine.js.

You can add and update Workflow Engine functionality. See Install Add-Ons for more information.

If you want to create your own Workflow Engine Moolet, see Create a Workflow Engine Moolet.

Manage Workflows

You can create and configure individual workflows or chains of workflows in the UI.Cisco Crosswork Situation Manager.

Read through Workflow Engine Strategies and Tips for ideas about how to use the Workflow Engine.

To access the Workflow Engine, navigate to Settings > Automation.

When you open the Workflow Engine, you see workflow tabs for the different types of engines you can enable at startup.

Click +Add Worfklow to create a workflow or double-click an existing workflow to open it.

Edit the Workflow Definition as follows to control which data the engine processes:

Workflow Property	Description
Workflow Name	Identifies the workflow.
Description	Describes the purpose of the workflow and what it should do.
Entry Filter	Identifies the criteria for events, alerts, or Situations to process with the current engine. Anything that does not meet the criteria skips to the next engine or Moolet in the chain. For example, you can set a filter on "severity > 3" to process only Major and Critical

	severity alerts.
Sweep Up Filter	Check the database for all objects that match the filter criteria and pass them to certain workflow actions as a list parameter. The sweep up filter expedites entry of related objects into the workflow. For example if you receive a link-up alert, you can set a filter to retrieve all related link-down alerts from the database and have the sweep up filter close them. The sweep up filter only applies to certain actions. You can see the actions that use the Sweep Up filter in the Workflow Engine Functions Reference and the topics for individual actions.Workflow Engine Functions Reference Note Cisco restricts use of the sweep up filter to open alerts that reside in the live database and does not permit the sweep up filter to pick up archived or closed alerts. Whether an archived or closed alert resides in the live database or elsewhere is immaterial, Cisco does not support this functionality and it is strictly prohibited.
First Match Only	Allow only the first occurrence of an object to pass through the workflow.

You can use the slider in the title bar to set the engine to Active or Inactive.

If you have multiple workflows, you can use the up and down arrows in Edit mode to reorder them. Alternatively, you can drag and drop the workflows into order.

After you create an engine, you can add actions to process data. When the engine is active, it processes all objects that match the filtering criteria according to the actions. If a user has manually changed an alert or Situation, this may affect its processing by the Workflow Engine.

Workflow Engine Functions Reference

This is a reference for Workflow Engine functions in Cisco Crosswork Situation Manager.

Functions may be available for more than one object. For example, addItemToList is available in event, alert, enrichment, and Situation workflows. In this reference, the functions appear in the lists for all the objects they are valid for.

Event functions

The following functions are available in event workflows:

- addDefaultValues: Adds a set of default values to **`custom_info`** based on a payload map. Sweep up filter applies.
- addItemToList: Adds an item or items to an array. Sweep up filter applies.
- addTags: Adds or updates a custom info field called "tags" with an array of string values.
- appendFields: Appends a concatenated set of fields to an existing field, using a separator character.
- appendString: Appends a static string to an existing field separated by a space character.
- ceventFilter: Returns **`true`** if the object matches a SQL-like filter. Sweep up filter applies.
- checkSeverity: Checks the severity level of the object.

- classifyEvent: Sets the class, type, and severity fields of an event based upon its contents using a predefined classification algorithm.
- concatFields: Sets the value of a field to a string representing a set of concatenated fields.
- contextFilter: Filters a **`workflowContext`** object for a specified name field. Sweep up filter applies.
- convertToJSON: Converts the object to JSON and adds it to the workflowContext for use in subsequent actions.
- copyFieldFromAlertToEvent: Copies a single field from an existing alert to a deduplicating event for the same alert.
- copyFromAlertToEvent: Copies multiple fields from an existing alert to a deduplicating event for the alert.
- copyFromContext: Copies a field from the **`workflowContext`** to a destination object field. Sweep up filter applies.
- copyToContext: Copies an object field to the **`workflowContext`**.
- copyToPayload: Copies a value to the payload in **`workflowContext`** for the current object.
- createPayload: Creates a **`workflowContext`** payload from the triggering object using a predefined payload map.
- deleteEnrichment:Removes data from the enrichment datastore.
- deltaEvent: Returns **`true`**: if the specified event fields differ from corresponding fields in an existing alert, or when an error occurs in the delta check, or when no alert exists. Returns **`false`** when it detects no changes.
- dnsLookup: Performs a lookup of an IP address or name to return a JSON object containing the IP address, FQDN, and name for the address.
- dropEvent: Allows you to prevent further processing of an event.
- estimateSeverity: Uses a predefined classification algorithm to estimate event or alert severity. Sweep up filter applies.
- existingAlertFilter: Returns **`true`** if the existing alert for a deduplicating event matches a SQL-like filter.
- getIntegrationConfig: Retrieves an integration configuration and stores it in the **`workflowContext`** for subsequent actions to use.
- getPayload: Creates a **`workflowContext`** payload from the triggering object from a predefined payload map. Sweep up filter applies.
- isClear: Returns **`true`** if the object's severity level is Clear (0).
- isInSubnet: Returns **`true`** when an IP address is present within a specified subnet. Sweep up filter applies.
- isNewerThan: Returns **`true`** when the object age in seconds is less than a specified age in seconds. Sweep up filter applies.
- isNotClear: Returns **`true`** if the object's severity level is not "Clear". Sweep up filter applies.
- isNotNull: Returns **`true`** if the value for an object's cEvent field is not null, is not an empty object, or is not an empty array.

- isNull: Returns **`true`** if the value for an object's cEvent field is null, is not set, is an empty object, or is an empty array.
- isOlderThan: Returns **`true`** when the object age in seconds is older than a specified age in seconds. Sweep up filter applies.
- listContains: Returns **`true`** when the array field you query contains some of your specified values. Sweep up filter applies.
- listContainsAll: Returns **`true`** when the array field you query contains all of your specified values. Sweep up filter applies.
- listDoesNotContain: Returns **`true`** when the array field you query contains none of your specified values. Sweep up filter applies.
- logCEvent: Prints a warning level message containing the current in-scope object in a readable JSON format to the Moogfarmd log file. Sweep up filter applies.
- logMessage: Logs a message to the Moogfarmd log.
- logWorkflowContext: Logs the contents of **`workflowContext`** to the current Moogfarmd log file at a warning level.
- logWorkflowDuration: Logs debug messages for the workflow execution duration.
- lowerCase: Changes the value of a field to lower case. Sweep up filter applies.
- populateNamedTopology: Populates the named topology field **`custom_info.moog_topology`** with a value. It can be a string value or the value of an alert attribute. Sweep up filter applies.
- prependFields: Prepends a concatenated set of fields to an existing field, using a separator character.
- prependString: Prepends a string to an existing field, using a separator character.
- restAsyncPost: Makes a HTTP POST request with a JSON payload to a named REST endpoint.
- searchAndReplace: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Sweep up filter applies.
- searchAndReplaceOrdered: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Allows you to provide the map as an array to preserve mapping order. Sweep up filter applies.
- sendToWorkflow; Sends the in-scope object to a named workflow in an Inform based workflow engine.
- setAgent: Sets the agent of the event or alert.
- setAgentLocation: Sets the agent location of the event or alert.
- setAgentTime: Sets the agent_time of the event to current time if the field does not exist in the event, or is more than the offset seconds in the past/future.
- setEnrichment: Updates a single record in the enrichment datastore with data from an alert.
- setEnrichmentBulk: Updates multiple records in the enrichment datastore with an array of data from an alert.

- setExternalId: Sets the external ID of the event or alert.
- setManager Sets the manager of the event or alerts.
- setSource Sets the source of the event or alert.
- setSourceId Sets the source ID of the event or alert.
- setCoreEventField: Sets a single core event field to a value.
- simpleLookup: Defines the lookup as two arrays of equal length. Sweep up filter applies.
- skip: Forwards an in-scope event, alert or Situation to the next chained moolet using the standard forwarding mechanism, and skips the rest of the workflows in the current engine.
- staticLookup: Searches for a **key** in a static lookup table, retrieves the corresponding value, and applies that value to a **field** in the object.
- #: Stops the workflow.
- stripFQDN: Splits a fully qualified domain name (FQDN) into a hostname/short name and a domain name and updates fields with the values.
- upperCase: Changes the value of a field to uppercase. Sweep up filter applies.
- willCreateNewAlert: Returns **true** if the event will create a new alert.
- willDeduplicateAlert: Returns **true** if the event will deduplicate into an existing alert.

Alert and enrichment functions

The following functions are available in alert and enrichment workflows:

- ackNotification: Automatically acknowledges a notification for a service.
- activateTopology: Updates a named topology from an inactive to an active state.
- addDefaultValues: Adds a set of default values to **custom_info** based on a payload map. Sweep up filter applies.
- addItemToList: Adds an item or items to an array. Sweep up filter applies.
- addTags: Adds or updates a custom info field called "tags" with an array of string values.
- addTopologyLink: Creates a link between two endpoints, A (source node) and Z (sink node), in a named topology.
- addTopologyNode: Creates a node in a named topology.
- alertDelta: Returns **true** when attributes have changed.
- alertInSituation: Returns **true** when the alert is a member of an active Situation. Sweep up filter applies.
- alertNotInSituation: Returns **true** when the alert is not a member of an active Situation. Sweep up filter applies.
- appendFields: Appends a concatenated set of fields to an existing field, using a separator character.
- appendString: Appends a static string to an existing field separated by a space character.
- assignAlert: Assigns an owner of in-scope alerts. Sweep up filter applies.

- between: Returns **`true`** if the object creation date falls between two times.
- ceventFilter: Returns **`true`** if the object matches a SQL-like filter. Sweep up filter applies.
- checkSeverity: Checks the severity level of the object.
- checkTopology: Checks for the existence of a named topology.
- checkTopologyLink: Checks for a link between two endpoints, A (source node) and Z (sink node), in a named topology.
- cloneTopology: Copies an existing topology to a new inactive named topology if the name is not already in use.
- closeAlert: Closes alerts.
- concatFields: Sets the value of a field to a string representing a set of concatenated fields.
- contextFilter: Filters a **`workflowContext`** object for a specified name field. Sweep up filter applies.
- convertToJSON: Converts the object to JSON and adds it to the workflowContext for use in subsequent actions.
- copyFromContext: Copies a field from the **`workflowContext`** to a destination object field. Sweep up filter applies.
- copyToContext: Copies an object field to the **`workflowContext`**.
- copyToPayload: Copies a value to the payload in **`workflowContext`** for the current object.
- createNotification: Automatically creates a notification for a service.
- createPayload: Creates a **`workflowContext`** payload from the triggering object using a predefined payload map.
- createTopology: Creates a named topology if it does not already exist. Takes no action if the topology exists.
- deactivateTopology: Updates a named topology from an active to an inactive state.
- deassignAlert: Removes the current owner of in-scope alerts. Sweep up filter applies.
- deleteEnrichment:Removes data from the enrichment datastore.
- deleteTopology: Delete a named topology
- deleteTopologyLink: Removes a direct link between two endpoints, A (source node) and Z (sink node), in a named topology.
- deleteTopologyNode: Deletes a node in a named topology.
- dnsLookup: Performs a lookup of an IP address or name to return a JSON object containing the IP address, FQDN, and name for the address.
- doesNotHaveStatus: Returns **`true`** when the in-cope alert or Situation is not in any of the specified states.

- estimateSeverity: Uses a predefined classification algorithm to estimate event or alert severity. Sweep up filter applies.
- exportViaKafka: Exports the payload from a createPayload to an external Kafka endpoint. Sweep up filter applies.
- exportViaRest: Exports the payload from a createPayload to an external REST endpoint. Sweep up filter applies.
- forward: Forwards the object to the named Moolet.
- getEnrichment: Retrieves data from the enrichment datastore through the Cisco Crosswork Situation Manager Enrichment API. Sweep up filter applies.
- getPayload: Creates a **`workflowContext`** payload from the triggering object from a predefined payload map. Sweep up filter applies.
- getIntegrationConfig: Retrieves an integration configuration and stores it in the **`workflowContext`** for subsequent actions to use.
- hasStatus: Returns **`true`** when the in-scope alert or Situation is in any of the specified states.
- isAssigned: Returns **`true`** if the object has an owner or moderator. Sweep up filter applies.
- isClear: Returns **`true`** if the object's severity level is Clear (0).
- isInSubnet: Returns **`true`** when an IP address is present within a specified subnet. Sweep up filter applies.
- isNewerThan: Returns **`true`** when the object age in seconds is less than a specified age in seconds. Sweep up filter applies.
- isNotAssigned: Returns **`true`** if the object does not have an owner or moderator. Sweep up filter applies.
- isNotClear: Returns **`true`** if the object's severity level is not "Clear". Sweep up filter applies.
- isNotNull: Returns **`true`** if the value for an object's cEvent field is not null, is not an empty object, or is not an empty array.
- isNull: Returns **`true`** if the value for an object's cEvent field is null, is not set, is an empty object, or is an empty array.
- isOlderThan: Returns **`true`** when the object age in seconds is older than a specified age in seconds. Sweep up filter applies.
- listContains: Returns **`true`** when the array field you query contains some of your specified values. Sweep up filter applies.
- listContainsAll: Returns **`true`** when the array field you query contains all of your specified values. Sweep up filter applies.
- listDoesNotContain: Returns **`true`** when the array field you query contains none of your specified values. Sweep up filter applies.
- logCEvent: Prints a warning level message containing the current in-scope object in a readable JSON format to the Moogfarmd log file. Sweep up filter applies.
- logMessage: Logs a message to the Moogfarmd log.

- logWorkflowContext: Logs the contents of **`workflowContext`** to the current Moogfarmd log file at a warning level.
- logWorkflowDuration: Logs debug messages for the workflow execution duration.
- lookupAndReplace: Sets the **`alertField`** to a value when one of the fields in the **`inFields`** list matches a word or regular expression. Sweep up filter applies.
- lowerCase: Changes the value of a field to lower case. Sweep up filter applies.
- populateNamedTopology: Populates the named topology field **`custom_info.moog_topology`** with a value. It can be a string value or the value of an alert attribute. Sweep up filter applies.
- prependFields: Prepends a concatenated set of fields to an existing field, using a separator character.
- prependString: Prepends a string to an existing field, using a separator character.
- replaceString: Replaces a string or regular expression in a field with a specified string or regular expression.
- resolveNotification: Automatically resolves a notification for a service.
- restAsyncPost: Makes a HTTP POST request with a JSON payload to a named REST endpoint.
- searchAndReplace: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Sweep up filter applies.
- searchAndReplaceOrdered: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Allows you to provide the map as an array to preserve mapping order. Sweep up filter applies.
- sendMooletInform: Sends a Moolet inform with a subject and details.
- sendToWorkflow; Sends the in-scope object to a named workflow in an Inform based workflow engine.
- sendViaRest: Sends the payload from a createPayload to an external REST endpoint. Sweep up filter applies.
- setAgent: Sets the agent of the event or alert.
- setAgentLocation: Sets the agent location of the event or alert.
- setClass: Sets the class of the alert.
- setCustomInfoJSONValue: Adds or updates a custom info key to the specified JSON value. Sweep up filter applies.
- setCustomInfoValue: Adds or updates a custom info key to a specified string value. Sweep up filter applies.
- setDescription: Sets the description of the object.
- setEnrichment: Updates a single record in the enrichment datastore with data from an alert.
- setEnrichmentBulk: Updates multiple records in the enrichment datastore with an array of data from an alert.

- setExternalId: Sets the external ID of the event or alert.
- setManager Sets the manager of the event or alerts.
- setSource Sets the source of the event or alert.
- setSourceId Sets the source ID of the event or alert.
- setSeverity: Sets the severity of the alert. Sweep up filter applies.
- setType: Sets the type of the alert.
- simpleLookup: Defines the lookup as two arrays of equal length. Sweep up filter applies.
- skip: Forwards an in-scope event, alert or Situation to the next chained moolet using the standard forwarding mechanism, and skips the rest of the workflows in the current engine.
- staticLookup: Searches for a **`key`** in a static lookup table, retrieves the corresponding value, and applies that value to a **`field`** in the object.
- #: Stops the workflow.
- stripFQDN: Splits a fully qualified domain name (FQDN) into a hostname/short name and a domain name and updates fields with the values.
- upperCase: Changes the value of a field to uppercase. Sweep up filter applies.

Situation functions

The following functions are available in Situation workflows:

- ackNotification: Automatically acknowledges a notification for a service.
- addDefaultValues: Adds a set of default values to **`custom_info`** based on a payload map. Sweep up filter applies.
- addItemToList: Adds an item or items to an array. Sweep up filter applies.
- addTags: Adds or updates a custom info field called `"tags"` with an array of string values.
- addThreadEntry: Adds a post to the named thread in the Collaboration tab of the Situation Room.
- appendFields: Appends a concatenated set of fields to an existing field, using a separator character.
- appendString: Appends a static string to an existing field separated by a space character.
- between: Returns **`true`** if the object creation date falls between two times.
- ceventFilter: Returns **`true`** if the object matches a SQL-like filter. Sweep up filter applies.
- checkSeverity: Checks the severity level of the object.
- checkSituationFlag: Checks if a specific flag is set for a Situation.
- checkSituationState: Returns true if the specified state exists for a Situation. Sweep up filter applies.
- concatFields: Sets the value of a field to a string representing a set of concatenated fields.
- containsAlertDetails: Returns **`true`** if all or any of the alerts in the Situation matches the filter condition. Sweep up filter applies.
- contextFilter: Filters a **`workflowContext`** object for a specified name field. Sweep up filter applies.

- convertToJSON: Converts the object to JSON and adds it to the workflowContext for use in subsequent actions.
- copyFromContext: Copies a field from the **`workflowContext`** to a destination object field. Sweep up filter applies.
- copyToContext: Copies an object field to the **`workflowContext`**.
- copyToPayload: Copies a value to the payload in **`workflowContext`** for the current object.
- createNotification: Automatically creates a notification for a service.
- createPayload: Creates a **`workflowContext`** payload from the triggering object using a predefined payload map.
- dnsLookup: Performs a lookup of an IP address or name to return a JSON object containing the IP address, FQDN, and name for the address.
- exportViaKafka: Exports the payload from a createPayload to an external Kafka endpoint. Sweep up filter applies.
- exportViaRest: Exports the payload from a createPayload to an external REST endpoint. Sweep up filter applies.
- createServiceTicket: Creates a ticket for the specified service.
- doesNotHaveStatus: Returns **`true`** when the in-cope alert or Situation is not in any of the specified states.
- filterByCookbook: Returns **`true`** if the Visualize data for the Situation matches the cookbook name.
- filterByCookbookAndRecipe: Returns **`true`** if the Visualize data for the Situation matches the cookbook name and recipe name.
- filterByRecipe: Returns **`true`** if the Visualize data for the Situation matches the recipe name.
- forward: Forwards the object to the named Moolet.
- getIntegrationConfig: Retrieves an integration configuration and stores it in the **`workflowContext`** for subsequent actions to use.
- getPayload: Creates a **`workflowContext`** payload from the triggering object from a predefined payload map. Sweep up filter applies.
- getSituationFlags: Retrieves the Situation flags and stores them in the workflowContext for subsequent actions to use.
- getVisualizationData: Retrieves the Visualize data and stores them in the workflowContext for subsequent actions to use.
- hasCausalPRC: Returns **`true`** if one or more alerts in the Situation has a causal PRC flag set. Sweep up filter applies.
- hasMerged: Returns **`true`** if the Situation has been merged or superseded.
- hasNotMerged: Returns **`true`** if the Situation has not been merged or superseded.

- hasSimilarSituations: Returns **true** when the Situation has a similar Situation above the specified threshold.
- hasStatus: Returns **true** when the in-scope alert or Situation is in any of the specified states.
- isAlertAcknowledged: Returns **true** when the in-scope alert state is Acknowledged.
- isAlertNotAcknowledged: Returns **true** when the in-scope alert state is not Acknowledged.
- isAssigned: Returns **true** if the object has an owner or moderator. Sweep up filter applies.
- isClear: Returns **true** if the object's severity level is Clear (0).
- isNotAssigned: Returns **true** if the object does not have an owner or moderator. Sweep up filter applies.
- isNewerThan: Returns **true** when the object age in seconds is less than a specified age in seconds. Sweep up filter applies.
- isNotClear: Returns **true** if the object's severity level is not "Clear". Sweep up filter applies.
- isNotNull: Returns **true** if the value for an object's cEvent field is not null, is not an empty object, or is not an empty array.
- isNull: Returns **true** if the value for an object's cEvent field is null, is not set, is an empty object, or is an empty array.
- isOlderThan: Returns **true** when the object age in seconds is older than a specified age in seconds. Sweep up filter applies.
- labelSituation: Labels the Situation using the Situation Manager Labeler macro language. Sweep up filter applies.
- listContains: Returns **true** when the array field you query contains some of your specified values. Sweep up filter applies.
- listContainsAll: Returns **true** when the array field you query contains all of your specified values. Sweep up filter applies.
- listDoesNotContain: Returns **true** when the array field you query contains none of your specified values. Sweep up filter applies.
- logCEvent: Prints a warning level message containing the current in-scope object in a readable JSON format to the Moogfarmd log file. Sweep up filter applies.
- logMessage: Logs a message to the Moogfarmd log.
- logWorkflowContext: Logs the contents of **workflowContext** to the current Moogfarmd log file at a warning level.
- logWorkflowDuration: Logs debug messages for the workflow execution duration.
- lowerCase: Changes the value of a field to lower case. Sweep up filter applies.
- prependFields: Prepends a concatenated set of fields to an existing field, using a separator character.
- prependString: Prepends a string to an existing field, using a separator character.
- removeSituationFlag: Removes a specific flag from a Situation.

- replaceString: Replaces a string or regular expression in a field with a specified string or regular expression.
- resolveNotification: Automatically resolves a notification for a service.
- resolveSituation: Marks in-scope Situations as Resolved if they match the workflow's entry filter and sweep up filter.
- reviveSituation: Revives (sets to Open) a Situation that is currently set to Resolved.
- restAsyncPost: Makes a HTTP POST request with a JSON payload to a named REST endpoint.
- searchAndReplace: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Sweep up filter applies.
- searchAndReplaceOrdered: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Allows you to provide the map as an array to preserve mapping order. Sweep up filter applies.
- sendMooletInform: Sends a Moolet inform with a subject and details.
- sendToWorkflow; Sends the in-scope object to a named workflow in an Inform based workflow engine.
- sendViaRest: Sends the payload from a createPayload to an external REST endpoint. Sweep up filter applies.
- setCustomInfoJSONValue: Adds or updates a custom info key to the specified JSON value. Sweep up filter applies.
- setCustomInfoValue: Adds or updates a custom info key to a specified string value. Sweep up filter applies.
- setDescription: Sets the description of the object.
- setSituationFlag: Sets a flag for a Situation.
- sigActionFilter: Returns **`true`** if the Situation action is of the specified type.
- sigActionToolFilter: Returns **`true`** if the specified tool has been run against a Situation.
- simpleLookup: Defines the lookup as two arrays of equal length. Sweep up filter applies.
- situationDelta Returns **`true`** when attributes have changed.
- skip: Forwards an in-scope event, alert or Situation to the next chained moolet using the standard forwarding mechanism, and skips the rest of the workflows in the current engine.
- staticLookup: Searches for a **`key`** in a static lookup table, retrieves the corresponding value, and applies that value to a **`field`** in the object.
- #: Stops the workflow.
- upperCase: Changes the value of a field to uppercase. Sweep up filter applies.

Infrastructure and Automation functions

The following functions are available in specific infrastructure and automation workflows:

- getJDBCEnrichment: Adds data to alerts from a JDBC database. Available in JDBC Enrichment workflows.
- getServiceNowEnrichment: Adds data to alerts from a ServiceNow database.
- sendToAnsible: Sends an automation request to Ansible. Available in Ansible Alert and Ansible Situation workflows.
- sendToAutomation: Sends an automation request. Available in EyeShare Alert, EyeShare Situation, Ignio Alert, and Ignio Situation workflows.
- sendToPuppet: Sends an automation request to Puppet. Available in Puppet Alert and Puppet Situation workflows.
- setAnsibleJob: Sets the instance and job template rule to use for Ansible automation requests. Available in Ansible Alert and Ansible Situation workflows.
- setAutomationPayload: Sets the automation solution, instance and Workflow Payload rule set to use for automation requests. Available in EyeShare Alert, EyeShare Situation, Ignio Alert, and Ignio Situation workflows.
- setPuppetAutomation: Sets the instance and job template rule to use for Puppet automation requests. Available in Puppet Alert and Puppet Situation workflows.

ackNotification

A Workflow Engine function that automatically acknowledges a notification for a service.

This function currently supports the PagerDuty and OpsGenie integrations.

This function is available as a feature of 7.4 integrations.

This function requires you to have already configured the services you want to use it with. When you configure some integrations, Cisco Crosswork Situation Manager automatically creates a workflow with the createNotification function; ensure that this workflow is active before you configure the **`ackNotification`** function. Integrations this function applies to indicate their compatibility on the UI.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`ackNotification`** takes the following arguments:

Name	Required	Type	Description
`services`	Yes	String	Comma separated list of the service names.

Example

The following example demonstrates typical use of Workflow Engine function **`ackNotification`**.

After you have configured the PagerDuty integration, you can configure a workflow with this function to automatically acknowledge alerts or Situations that Cisco Crosswork Situation Manager sends to PagerDuty.

- **`services`**: PagerDuty

The UI translates this setting to the following JSON:

```
{"services":"PagerDuty"}
```

Now when Cisco Crosswork Situation Manager sends alert or Situation data to PagerDuty, the corresponding PagerDuty incident is automatically set to `"Acknowledged"`.

activateTopology

A Workflow Engine function that updates a named topology from an inactive to an active state. Returns false if the topology can not be activated.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`activateTopology`** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology)

Example

The following example demonstrates typical use of Workflow Engine function **`activateToplogy`**.

If you want to activate an inactive topology named `"my network"`, set the following:

- **`topologyName`**: my network

The UI translates your settings to the following JSON:

```
{"topologyName":"my network"}
```

If you run the **`topologies`** API, you can see your new topology:

```
curl -X GET 'https://example.com/api/v1/topologies'
```

Returns the following:

```
{
   "name": "my network",
   "active": true,
   "description": "my network nodes"
}
```

addDefaultValues

A Workflow Engine function that adds a set of default values to **`custom_info`** based on a payload map.

You can use this function as part of an Alert Enrichment engine, where it precedes any dynamic enrichment. The map can contain plain text, substitutions (for example, **`$severity`**, **`$custom_info.a.b.c.d`**) and complex objects (for example, **`{ "country" : "Unknown", "city" : "Unknown" }`**). See Payload Maps to learn how to define maps.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, enrichment, and Situation workflows. At alert level this function can either run in an Event Workflow Engine, or an Alert Workflow Engine, the choice of which depends on what else happens to the data (for example, whether it is further added to, or overwritten), and if **`custom_info`** is de-duplicated as part of the alert creation process (by default it is not).

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`addDefaultValues`** takes the following arguments:

Name	Required	Type	Description
`mapName`	Yes	String	Name of the map defined in the PayloadMaps integration.
`key`	Yes	String	Destination **`custom_info`** location.

Example

The following example demonstrates typical use of Workflow Engine function **`addDefaultValues`**.

You have created a payload map called `"Default Enrichment"`. You can now create a workflow to add the resulting map to a CEvent object before enrichment takes place (ensuring that the object has a set of populated values). To hold your enrichment data in **`custom_info.enrichment.myCMDB`**, set the following:

- **`mapName`**: DefaultEnrichment
- **`key`**: custom_info.enrichment.myCMDB

The UI translates your settings to the following JSON:

```
{"mapName":"DefaultEnrichment","key":"custom_info.enrichment.myCMDB"}
```

Note

This function does not store the resulting map in **`workflowContext`**, and so the result of this action is not available to subsequent actions.

addItemToList

A Workflow Engine function that adds an item or items to an array. You can specify more than one value. Does not add duplicate elements to the array, but maintains an array of unique elements. Returns an array of unique items.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`addItemToList`** takes the following arguments:

Name	Required	Type	Description
`field`	Yes	String	Name of the field to add the elements to. If the field does not exist, creates it.
`items`	Yes	Object	Values of the items to add as an array.

addTags

A Workflow Engine function that adds or updates a custom info field called “tags” with an array of string values.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **addTags** takes the following arguments:

Name	Required	Type	Description
tags	Yes	Object	Array of tags to add. For example, ["tag1", "tag2"]

Example

The following example demonstrates typical use of Workflow Engine function **addTags**.

To add the tags “traps” and “network” to alerts of SNMP traps of networking devices, set the following:

- **tags**: ["traps", "network"]

The UI translates your settings to the following JSON:

```
{"tags":"["traps", "network"]"}
```

If successful, the function returns **true** and adds the tags to in-scope alerts under **custom_info**. You can also now use the Tag field in UI filters New tags merge with those already in the **custom_info.tags** field. For example, if there are existing tags, and **custom_info.tags** is therefore present:

```
{ "tags":["snmp"]}
```

The new tags now also appear in this field:

```
{ "tags":["snmp", "traps", "network"]}
```

addThreadEntry

A Workflow Engine function that adds a post to the named thread in the Collaboration tab of the Situation Room. Defaults to the Support thread.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **addThreadEntry** takes the following arguments:

Name	Required	Type	Description

entry	Yes	String	Thread entry text to add.
threadName	No	String	Name of the thread. Defaults to Support.

Example

The following example demonstrates typical use of Workflow Engine function **addThreadEntry**.

To create a new thread for a Situation, set the following:

- **entry**: New Entry
- **thread_Name**: Thread 0958

The UI translates your settings to the following JSON:

```
{"entry":"New Thread","threadName":"Thread 0958"}
```

addTopologyLink

A Workflow Engine function that attempts to create a link between two endpoints, A (source node) and Z (sink node), in a named topology. Creates and activates the named topology if it does not exist. Leaves existing inactive topologies inactive. Adds the referenced nodes if they do not exist.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **addTopologyLink** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology)
sourceNode	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a value, use $(<attribute_name>). For example $(custom_info.mySourceNode)
sinkNode	yes	string	The name or substituted value for the 'Z' endpoint (sink node). To substitute a value, use $(<;attribute_name>). For example $(custom_info.mySinkNode)
description	no	string	Optional link description. When not supplied, defaults to the time, date, and the triggering alert id.

Example

The following example demonstrates typical use of Workflow Engine function **addTopologyLink**.

If you want to create a link in the topology "My Network" between the alert source and another node you have previously added to the workflow context, set the following:

1. **topologyName**: my network
2. **sourceNode**: $(source)
3. **sinkNode**: $(workflowContext.destination)

The UI translates your settings to the following JSON:

```
{"topologyName":"my network",
"sourceNode":"$(source)","sinkNode":"$(workflowContext.destination)"}
```

For an alert where **source** = sflinux101 and the corresponding **workflowContext.destination** = sflinux102, you can run the **topologies** API to see your new link:

```
curl -X GET 'https//example.com/api/v1/topologies/my%20network/links'
```

Returns the following:

```
[{
    "description": "Automatically created link:
triggering alert # 35 @ 2020-05-08T02:14:05.680Z",
    "sourceNode": "sflinux101",
    "sinkNode": "sflinux102"
}]
```

addTopologyNode

A Workflow Engine function that creates a node in the named topology. The node name can be static or a substitution value. To substitute a value, use $(<attribute_name>). For example $(source). Creates the named topology if the it does not exist. As a best practice, set 'first match only' for this action.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **addTopologyNode** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology).
nodeName	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a value, use $(<attribute_name>). For example $(custom_info.node).
description	no	string	Optional node description. When not supplied, defaults to the time, date, and the triggering alert id.

Example

The following example demonstrates typical use of Workflow Engine function **addTopologyNode**.

If you want to create a node in the topology "My Network" for the alert source, set the following:

- **topologyName**: my network
-)**sourceNode**: $(source)

The UI translates your settings to the following JSON:

```
{"topologyName":"my network","nodeName":"$(source)"}
```

For an alert where **source** = sflinux101, you can run the **topologies** API to see your new node:

```
curl -X GET 'https//example.com/api/v1/topologies/my%20network/nodes    '
```

Returns the following:

```
[{   "name": "sflinux101",
"description": "Automatically created node: triggering alert # 35 @ 2020-05-
08T02:14:05.680Z"  }]
```

alertDelta

A Workflow Engine function that returns **`true`** when attributes have changed. This is based on the **`previous_data`** metadata, which Cisco Crosswork Situation Manager sends with the alert object in an AlertUpdate event.

Only use this function in conjunction with an entry filter that includes the **`event_handler`** trigger for "Alert Updated".

This function does not check the values of the attributes, only if the attributes have changed. As standard de-duplication changes attributes, use this function carefully.

Cisco recommends placing **`alertDelta`** in an engine dedicated to handling Alert Updates and other alert event handlers. This prevents updated alerts re-entering the processing chain through standard Alert Workflows. Contact your Cisco Crosswork Situation Manager administrator for more information.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for alert and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`alertDelta`** takes the following arguments:

Name	Required	Type	Description
`fields`	Yes	Object	List of attributes to check for change. Accepts granular custom info attributes.

Example

The following example demonstrates typical use of Workflow Engine function **`alertDelta`**.

You want to check if the owner an alert has changed before performing subsequent actions in your workflow. You could use an entry filter to check for a specific ownership, but in this instance the value of the ownership is not relevant, only that it has changed.

Using a separate Workflow Engine to prevent unwanted re-entry, you set up a workflow with an entry filter that includes the **`event_handler`** trigger for "Alert Update" and the owner as "Unassigned":

(event_handler = "Alert Update") AND (owner != "anon")

Set the following:

- **`fields`**: owner
- Forwarding behavior: Stop this workflow. This ensures that if the alert owner has not changed, subsequent actions in this workflow do not execute.

The UI translates your settings to the following JSON:

```
{"fields":["owner"]}
```

If the alerts metadata shows that the "owner" has changed, the function returns **`true`** and the alert is forwarded to the next action in the workflow.

If function does not detect a change of ownership, the function returns **`false`** and the forwarding behaviour prevents subsequent actions in the workflow from executing.

alertInSituation

A Workflow Engine function that returns true when the alert is a member of an active Situation.

This function is available for alert and enrichment workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`alertInSituation`** has no arguments.

alertNotInSituation

A Workflow Engine function that returns true when the alert is not a member of an active Situation.

This function is available for alert and enrichment workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`alertNotInSituation`** has no arguments.

appendFields

A Workflow Engine function that appends a concatenated set of fields to an existing field, using a separator character.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`appendFields`** takes the following arguments:

Name	Required	Type	Description
`sourceFields`	Yes	object	An array of fields to concatenate, in the format: **`[ field1, ..., fieldn ]`**.
`separator`	Yes	string	Separator to use between the concatenated values. Do not use quotes around the separator.
`destination`	Yes	string	Destination field for the concatenated string.

appendString

A Workflow Engine function that appends a static string to an existing field separated by a space character.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **appendString** takes the following arguments:

Name	Required	Type	Description
string	Yes	string	String to append.
destination	Yes	string	Field to append the concatenated string to.

assignAlert

A Workflow Engine function that assigns an owner of in-scope alerts. Returns **true** if the function assigns at least one alert.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is available for alert and enrichment workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **assignAlert** takes the following arguments:

Name	Required	Type	Description
user	Yes	String	ID or username of the user to assign as owner. To specify a **workflowContext** key, prefix with "workflowContext." For example, "workflowContext.username"

Example

The following example demonstrates typical use of Workflow Engine function **assignAlert**.

To assign all in-scope alerts to the user "support", Set the following:

- **user**: support

The UI translates your settings to the following JSON:

```
{"user":"support"}
```

A more likely scenario is where a previous action has identified an appropriate user and populated a workflowContext key, “username”. Here, the following arguments assign the in-scope alerts to the username stored in that key:

```
{"user":"workflowContext.username"}
```

In either scenario, if the function is able to assign at least one alert to the user, it returns **true**.

between

A Workflow Engine function that returns true if the object creation date falls between two times. Optionally between times on specific days.

This function is available for alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`between`** takes the following arguments:

Name	Required	Type	Description
from	Yes	String	Start time in hh:mm:ss 24hr format.
to	Yes	String	End time in hh:mm:ss 24hr format.
days	Yes	Object	Optional array of days in short form: `"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"`. Use a blank array for all days.

Example

The following example demonstrates typical use of Workflow Engine function **`between`**. If you want to check for objects created at any time on Monday or on Friday, set the following:

- **`from`**: 00:00:00
- **`to`**: 24:00:00
- **`days`**: `["Mon","Fri"]`

The UI translates your settings to the following JSON:

```
{"from":"00:00:00","to":"24:00:00","days":["Mon","Fri"]}
```

ceventFilter

A Workflow Engine function that returns true if the object matches a SQL-like filter. This function uses the CEvents API to evaluate the SQL-like filter against the object.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`ceventFilter`** takes the following arguments:

Name	Required	Type	Description
`filter`	Yes	String	SQL-like filter expression. Use single quotes around strings.

			For example: **`description matches 'abc'`**

checkSeverity

A Workflow Engine function that checks the severity level of the object, either as a static value, or with an operator. Returns **`true`** if the severity level matches your specified value.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`checkSeverity`** takes the following arguments:

Name	Required	Type	Description
`severity`	Yes	Validated String	Severity value and optional operator. Enter a severity value between **`0`** and **`5`**. Valid operators are: **`>`**, **`<`**, **`>=`**, **`<=`**, **`!=`**, **`=`**.

Example

The following examples demonstrate typical use of Workflow Engine function **`checkSeverity`**.

If you want to check if an object has a severity level of less than 3, enter the following:

- **`severity`**: <3

The UI translates your settings to the following JSON:

```
{"severity":"<3"}
```

Given an object with the following data:

```
"severity": 1
```

The function returns true, since the severity level is less than 3.

Now consider an object with the following:

```
"severity": 3
```

In this case the function returns **`false`**, since the severity level is not less than 3.

However, if you were to check if the object has a severity level of less than or equal to 3 (**`<=3`**), the UI would translate your settings as follows:

```
{"severity": "<=3"}
```

For the same object, the function now returns **`true`**, since your criteria now includes 3, whereas before your criteria only included values less than 3.

checkSituationFlag

A Workflow Engine function that returns **`true`** if the specified flag is set for a Situation. For example TICKETING, or LEAVE_MANUAL_DESCRIPTION.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **checkSituationFlag** takes the following arguments:

Name	Required	Type	Description
flag	Yes	String	Name of the flag to check for.

Example

The following example demonstrates typical use of Workflow Engine function **checkSituationFlag**.

If you want to check if a Situation's flag is "TICKETED", enter the following:

- **flag**: TICKETED

The UI translates your settings to the following JSON:

```
{"flag":"TICKETED"}
```

Given a Situation with the following flag:

```
{"situationFlags": ["TICKETED"]}
```

The function returns **true**, since the object's flag matches "TICKETED".

Now consider this Situation:

```
{"situationFlags": ["TICKET_PENDING"]}
```

In this case the function returns **false**, since the Situation's flag does not match "TICKETED".

As a subsequent action you can set the flag using setSituationFlag.

checkSituationState

A Workflow Engine function that returns true if the specified state exists for a Situation. For example TICKETED, LEAVE_MANUAL_DESCRIPTION. Not to be confused with Situation status.

This function is available for Situation workflows only.

This function is only available as a feature of the Workflow Engine v1.0 download. The Workflow Engine v.1.1 replaces this function with checkSituationFlag.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **checkSituationState** takes the following arguments:

Name	Required	Type	Description
state	Yes	String	Situaton state or flag to check for. For example: "TICKETED".

Example

The following example demonstrates typical use of Workflow Engine function **checkSituationState**. If you want to set the state to TICKETED, enter the following:

- state: TICKETED

The UI translates your settings to the following JSON:

```
{"state":"TICKETED"}
```

Returns **true** for Situations that have a state of "TICKETED". For example a Situation 22 which returns the following for getSituationFlags :

```
{"22": ["TICKETED"]}
```

checkTopology

A Workflow Engine function that checks for the existence of a named topology. Returns true if the topology exists.

The topology name can be static or a substitution value. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology).

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **checkTopology** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology)

Example

The following example demonstrates typical use of Workflow Engine function **checkTopology**.

If you want to delete the topology stored in **custom_info.moog_topology** set the following:

- **topologyName**: $(custom_info.myTopology)

The UI translates your settings to the following JSON:

```
{"topologyName":"$(custom_info.myTopology)"}
```

For an alert with the following **custom_info.myTopology** = "my network", the action returns true if the "my network" topology exists.

checkTopologyLink

A Workflow Engine function that returns true when a link exists between two endpoints, A (source node) and Z (sink node), in a named topology. Assumes direct connection when no hop count is supplied. For hop counts greater than 1, returns true when the a connection exists and the distance between nodes is less than or equal to the hop count. Otherwise returns false.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **checkTopologyLink** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology)
sourceNode	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a value, use $(<attribute_name>). For example $(custom_info.mySourceNode)
sinkNode	yes	string	The name or substituted value for the 'Z' endpoint (sink node). To substitute a value, use $(<attribute_name>). For example $(custom_info.mySinkNode)
hopCount	no	string	Optional distance in hops between the nodes. Defaults to 1 when not specified.

Example

The following example demonstrates typical use of Workflow Engine function **checkTopologyLink**.

If you want to check for direct link in the topology "my network" between the alert source and another node you have previously added to the workflow context, set the following:

- **topologyName**: my network
- **sourceNode**: $(source)
- **sinkNode**: $(workflowContext.destination)

The UI translates your settings to the following JSON:

```
{"topologyName":"my network",
"sourceNode":"$(source)","sinkNode":"$(workflowContext.destination)"}
```

For an alert where **source** = sflinux101 and the corresponding **workflowContext.destination** = sflinux102, the function returns true when a direct link exists as follows:

```
[{
    "description": "Automatically created link:
triggering alert # 35 @ 2020-05-08T02:14:05.680Z",
    "sourceNode": "sflinux101",
    "sinkNode": "sflinux102"
}]
```

classifyEvent

A Workflow Engine function that sets the class, type, and severity fields of an event based upon its contents using a predefined classification algorithm. Overwrites existing values. See Event and Alert Field Best Practice for information on object fields.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`classifyEvent`** takes the following arguments:

Name	Required	Type	Description
`eventFields`	Yes	Object	An array of fields to use in the classification. An empty list, [], uses the description field.
`typeField`	Yes	String	Field to populate with the calculated 'type'.
`classField`	Yes	String	Field to populate with the calculated 'class'.
`severityField`	Yes	String	Field to populate with the calculated 'severity'.

Example

The following example demonstrates typical use of Workflow Engine function **`classifyEvent`**. If you want the Workflow Engine to automatically populate the type, class, and severity fields of the event based upon the description, set the following:

- eventFields: []
- typeField: type
- classField: class
- severityField: severity

The UI translates your settings to the following JSON:

```
{"eventFields":[],"typeField":"type","classField":"class","severityField":"sever
ity"}
```

Given an object with the following description:

```
"description":"App server APPSERVER2002 down."
```

The Workflow Engine updates the object fields as follows:

```
"severity": 5,
"type": "availability",
"class": "server"
```

cloneTopology

A Workflow Engine function that copies an existing topology to a new inactive named topology if the name is not already in use. Both topologyName and cloneName can be static or a substitution value. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology).

Follow this action with the 'activateTopology' action if you want to activate the new topology.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`cloneTopology`** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The 'donor' named topology or substituted value for the topology. May be active or inactive. To substitute a value, use $(<attribute_name>).

			For example $(custom_info.myTopology).
cloneName	yes	string	The 'clone' named topology or substituted value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology).

Example

The following example demonstrates typical use of Workflow Engine function **`cloneTopology`**.

If you want to create a new topology based upon a topology in the alert workflow context, set the following:

- **`topologyName`**: $(workflowContext.myTopology)
- **`cloneName`**: $(workflowContext.myTopology) copy

The UI translates your settings to the following JSON:

```
{"topologyName":"$(workflowContext.myTopology)","cloneName":"$(workflowContext.myTopology) copy"}
```

For an alert that has a **`workflowContext.myTopolgy`** = "my network", the action clones the topology to "my network copy". If you run the **`topologies`** API, you can see your new inactive topology:

```
curl -X GET 'https://example.com/api/v1/topologies/inactive'
```

Returns the following:

```
 [{
        "name": "my network copy",
        "active": false,
        "description": "Automatically created topology:
triggering alert # 35 @ 2020-05-08T02:14:05.680Z"
}]
```

closeAlert

A Workflow Engine function that closes alerts.

- As a best practice, it is better to use Auto Close rules to close alerts instead of the Workflow Engine. When possible, you should use the Event Workflows engine to prevent unnecessary alert creation. This way you can avoid creating an alert and immediately closing it during enrichment.
- You cannot modify a closed alert. If this function is not the last in a workflow, any subsequent functions that attempt to modify the alert may cause the workflow to fail.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for alert and enrichment workflows.

The workflow sweep up filter applies to this function. If the workflow has a sweep up filter, the function closes alerts the filter finds too so that other moolets can process them. For example, if you export alerts for reporting purposes.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`closeAlert`** has no arguments.

concatFields

A Workflow Engine function that sets the value of a field to a string representing a set of concatenated fields.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`concatFields`** takes the following arguments:

Name	Required	Type	Description
`sourceFields`	Yes	Object	An array of fields to concatenate, in the format: **`[ "field1", ..., "fieldn" ]`**.
`separator`	Yes	String	Separator to use between fields in concatenation. Do not quote the separator.
`destination`	Yes	String	Field to populate with the concatenated string.

containsAlertDetails

A Workflow Engine function that returns **`true`** if all or any of the alerts in the Situation matches the filter condition. Uses SQL-like filter syntax.

Applies the scope to all Situations in the Workflow when there are multiple Situations in context. For example, if you used a sweep up filter in the workflow definition. In this case, if you have set the scope to 'any', every Situation must have at least one alert match the SQL-like filter for the function to return true.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for Situation workflows only.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`containsAlertDetails`** takes the following arguments:

Name	Required	Type	Description
`scope`	Yes	String	Sets the scope of the contains match to: **`all`** : every alert within the Situation must match the SQL-like filter. **`any`**: at least one alert within the Situation must match the SQL-like filter Applies the scope to all Situations in the workflow.
`filter`	Yes	String	SQL-like CEvent filter to use to evaluate alerts against. For example: **`"severity > 1"`**.

Example

The following example demonstrates typical use of Workflow Engine function **`containsAlertDetails`**. If you want to verify that a Situation contains at least one severity 3 or higher alert, set the following:

- `scope`: any
- filter: severity >= 3

The UI translates your settings to the following JSON:

```
{"scope":"any","filter":"severity >= 3"}
```

contextFilter

A Workflow Engine function that filters a **`workflowContext`** object for a specified name field. Returns **`true`** if the function finds the field under the **`workflowContext`** object.

This function is available as a feature of the `Add-ons` v1.3 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`contextFilter`** takes the following arguments:

Name	Required	Type	Description
`field`	Yes	String	Name of the field to filter by.

Example

The following example demonstrates typical use of Workflow Engine function **`workflowEngineFunction`**.

You want to filter a **`workflowContext`** object for a field called `"visualize.my_cookbook"`. Set the following:

- **`field`**: visualize.my_cookbook

The UI translates your settings to the following JSON:

```
{ "field":"visualize.my_cookbook"}
```

If the `"visualize.my_cookbook"` field is present under the **`workflowContext`** object, the function returns **`true`**. If the field does not exist, the function returns **`false`**.

convertToJSON

A Workflow Engine function that converts the object to JSON and adds it to the workflowContext for use in subsequent actions such as the restAsyncPost function.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **convertToJSON** has no arguments.

copyFieldFromAlertToEvent

A Workflow Engine function that copies a single field from an existing alert to a deduplicating event for the same alert. For example, if an update event doesn't include a 'source' attribute, the 'copyFromAlertToEvent' copies the 'source' from the existing alert. If the copy fails, the event would have an empty 'source' field causing the Alert Builder to reject it. In this case create a subsequent action to check for the existence of 'source enables you to set a default source for the event if it is undefined.

When using this function, the following applies:

- You can specify both the source and destination fields. Choose the forwarding action for this carefully.
- If the process fails and the forwarding behavior is 'Stop All Workflows', the Workflow Engine drops the event.
- If you select 'Always Forward' there is a risk that the Alert Builder will reject an incomplete event. For example, if an update event does not include a 'source' property, this function copies the value for 'source' from the existing alert.
- If the copy fails, the event has an empty 'source' field causing the Alert Builder to reject it. In this case, create a subsequent action to check for the existence of 'source' and set a default source for the event if it is undefined.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **copyFieldFromAlertToEvent** takes the following arguments:

Name	Required	Type	Description
sourceField	Yes	String	The field in the existing alert to copy.
destinationField	Yes	String	Destination field in the event.

copyFromAlertToEvent

A Workflow Engine function that copies multiple fields from an existing alert to a deduplicating event for the alert.

When using this function, the following applies:

- You can backfill fields from the alert to the event. Choose the forwarding action for this carefully.
- Returns **true** when the event has no matching existing alert. The same behavior as if all specified fields copied successfully. To avoid this behavior, use another function to check for event deduplication before copyFromAlertToEvent.

- If the forwarding behavior is 'Always Forward', there is a risk that the Alert Builder may reject an incomplete event.
- Use subsequent actions to validate the event and add defaults to any backfilled event fields as needed. For example, if an update event does not include a 'source' attribute, you can use this function to copy 'source' from the existing alert. If the copy fails, the event would have an empty 'source' field causing the Alert Builder to reject it. In this case, create a subsequent action to check for the existence of 'source' which enables you to set a default source for the event if it is undefined.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`copyFromAlertToEvent`** takes the following arguments:

Name	Required	Type	Description
fields	Yes	Object	Array of fields to copy from the alert to the event.

copyFromContext

A Workflow Engine function that copies a field from the **`workflowContext`** to a destination object field. The function overwrites the destination field if it exists, and creates and populates it if it does not. Returns **`false`** if the **`workflowContext`** field does not exist.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`copyFromContext`** takes the following arguments:

Name	Required	Type	Description
`from`	Yes	String	Source **`workflowContext`** field.
`to`	Yes	String	Destination object field.

Example

The following example demonstrates typical use of Workflow Engine function **`workflowEngineFunction`**.

To copy the value of the **`workflowContext`** field “visualise.cookbook_name” to “custom_info.sourceCookbook”, set the following:

- **`from`**: visualize.cookbook_name

- **to**: custom_info.sourceCookbook

The UI translates your settings to the following JSON:

```
{ "from":"visualise.cookbook_name", "to":"custom_info.sourceCookbook"}
```

copyToContext

A Workflow Engine function that copies an object field to the **workflowContext**. Returns **false** if the object field is null or does not exist.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **copyToContext** takes the following arguments:

Name	Required	Type	Description
from	Yes	String	Source object field.
to	Yes	String	Destination **workflowContext** field.

Example

The following example demonstrates typical use of Workflow Engine function **workflowEngineFunction**.

To copy the value of the CEvent field "custom_info.location.city" to "location.city" in **workflowContext**, set the following:

- **from**: custom_info.location.city
- **to**: location.city

The UI translates your settings to the following JSON:

```
{ "from":"custom_info.location.city", "to":"location.city"}
```

copyToPayload

A Workflow Engine function that copies a value to the payload in **workflowContext** for the current object. This can be a specific value, or a substitution for an existing object, such as **$custom_info.myvalue**.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **copyToPayload** takes the following arguments:

Name	Required	Type	Description

payloadKey	Yes	String	Key in the payload to insert data.
value	Yes	String	Value to insert into **payloadKey**

Example

The following example demonstrates typical use of Workflow Engine function **copyToPayload**.

As part of an alert data export you have a basic payload map defined called "AlertExport", which generates a map with the following keys and values:

```
{
  "alertId": "$alert_id",
  "summary": "$description"
}
```

At export time, you want to include the location, but the data is stored in different places in different alerts: either **ci.location.city** or **ci.location.dc.** Additionally you want to add the current time to the alert.

The solution is as follows:

- You create two workflows: one to add data from **custom_info.location**, the other from **custom_info.datacentre**.
- You have already configured default values for these properties, so you can create entry filters based on them. For example, an entry filter of **custom_info.location.city != 'Unknown'** ensures you are only copying data from alerts that have **city** set to a non-default value.
- You use the **copyToPayload** function to copy from the correct key into the payload using the following actions:
- **createPayload** configured with the map name (AlertExport).
- **copyToPayload** for alerts within the city, with **payloadKey** set to **"location"** and **value** set to **$custom_info.location.city**.
- **copyToPayload** with **payloadKey** set to **"currentTime"** and **value** set to **$moog_now**.
- **exportViaRest** configured with the endpoint name.

This configuration ensures your payload contains the additional data and exports to an external REST endpoint. The final payload is a combination of your base payload and the additional keys you have added to it:

```
{
  alertId: 11,
  summary: 'Ping fail 10.0.0.1',
  currentTime: 1576154788,
  location: 'London'
}
```

The actions returns **true** if the data successfully copies to the payload. If the data did not successfully copy, or the function did not find the payload you specified, it returns **false**.

createNotification

A Workflow Engine function that automatically creates a notification for a service.

This function currently supports the PagerDuty and OpsGenie integrations.

This function is available as a feature of 7.4 integrations.

This function requires you to have already configured the services you want to use it with. When you configure some integrations, Cisco Crosswork Situation Manager creates a workflow with this function that automatically communicates new alert or Situation data to the service. Integrations this function applies to indicate their compatibility on the UI.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`createNotification`** takes the following arguments:

Name	Required	Type	Description
`services`	Yes	String	Comma separated list of the service names.

Example

The following example demonstrates typical use of Workflow Engine function **`createNotification`**.

When you configure the PagerDuty integration, Cisco Crosswork Situation Manager automatically creates a workflow with this function, and is configured as follows:

- **`services`**: PagerDuty

The UI translates this setting to the following JSON:

```
{"services":"PagerDuty"}
```

This allows Cisco Crosswork Situation Manager to automatically notify PagerDuty of new alert or Situation data and make a request to raise an incident.

createPayload

A Workflow Engine function that creates a **`workflowContext`** payload from the triggering object from a predefined payload map. For use in subsequent actions, for example exportViaRest or exportViaKafka.

This function relates directly to the payload maps from your Payload Maps integration.

This function is available as a feature of the Workflow Engine v1.2 and later. If you are using a newer version of the Workflow Engine, use getPayload instead.

This function is available for event, alert, enrichment, and Situation workflows.

This function does not modify the in-scope object.

The workflow sweep up filter applies to this function. Swept up objects have an entry under the **`workflowContext.payloads`** object corresponding to their associated ID.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **createPayload** takes the following arguments:

Name	Required	Type	Description
mapName	Yes	String	Name of the map from the Payload Maps

Example

The following example demonstrates typical use of Workflow Engine function **createPayload**.

If you wanted to create a payload for a map called "AlertExport", set the following:

- **mapName**: AlertExport

The UI translates your settings to the following JSON:

```
{ "mapName":"AlertExport"}
```

The function returns **true** when it finds a map and creates the payload. It stores the payload in the **workflowContext** payloads key. The function also assigns the payload an identifier for the in-scope object.

For example, a payload created for an alert with the ID #1234 stores in **workflowContext** as follows:

```
"workflowConext" : {
  "payloads" : {
      "1234" : { ... }
  }
}
```

Note

All functions must use the same structure for payloads. For alerts and Situations, use the object ID. For events, use the signature value.

createServiceTicket

A Workflow Engine function that creates a ticket for the specified service.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **createServiceTicket** takes the following arguments:

Name	Required	Type	Description
services	Yes	String	Comma separated list of the service names: ServiceNow, Remedy, Cherwell, Jira Service Desk, Jira Software.

createTopology

A Workflow Engine function that creates a named topology if it does not already exist. Takes no action if the topology exists.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`createTopology`** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology)
description	no	string	Optional topology description. When not supplied, defaults to the time, date, and the triggering alert id
active	no	string	True or false depending on whether you want the topology to be active. Defaults to 'true'.

Example

The following example demonstrates typical use of Workflow Engine function **`createTopology`**.

If you want to create an active topology named `"my network"`, set the following:

1. **`topologyName`**: my network
2. **`description`**: my network nodes

The UI translates your settings to the following JSON:

```
{"topologyName":"my network","description":"my network nodes"}
```

If you run the **`topologies`** API, you can see your new topology:

```
curl -X GET 'https://example.com/api/v1/topologies'
```

Returns the following:

```
 {
    "name": "my nework",
    "active": true,
    "description": "my network nodes"
 }
```

deactivateTopology

A Workflow Engine function that updates a named topology from an active to an inactive state. Returns false if the topology can not be dedactivated.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`deactivateTopology`** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology).

Example

The following example demonstrates typical use of Workflow Engine function **`deactivateToplogy`**.

If you want to deactivate an active topology named "my network", set the following:

1. **topologyName**: my network

The UI translates your settings to the following JSON:

```
{"topologyName":"my network"}
```

If you run the **topologies** API, you can see your inactive topology:

```
curl -X GET 'https://example.com/api/v1/topologies/inactive'
```

Returns the following:

```
{
   "name": "my nework",
   "active": false,
   "description": "My network nodes"
}
```

deassignAlert

A Workflow Engine function that removes the current owner of in-scope alerts. Returns **true** if the function deassigns at least one alert.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is available for alert and enrichment workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **deassignAlert** has no arguments.

deleteEnrichment

A Workflow Engine function that removes data from the enrichment datastore. Returns **true** if the request is successful.

This function relates directly to the API details from your Enrichment API.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for event, alert, and enrichment workflows.

This function does not modify the in-scope object when it deletes enrichment data.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **deleteEnrichment** takes the following arguments:

Name	Required	Type	Description
attribute	Yes	String	Name of the attribute to lookup. For example, "hostname".

value	Yes	String	Name of the field or **workflowContext** key holding the data to lookup. To specify a **workflowContext** key, prefix with "workflowContext". For example, "workflowContext.lookupkey".

Wildcard Search

You can perform a wildcard search if the event field or **workflowContext** contains a wildcard string. Valid wildcards are as follows:

Wildcard Type	Example	Description
Ends with	"*searchString"	Matches strings ending with the string "searchString".
Begins with	"searchString*"	Matches strings beginning with the string "searchString".
Contains	"*searchString*"	Matches strings containing the string "searchString".

The following conditions apply when performing a wildcard search on an attribute or value:

Attribute Wildcard

The only supported wildcard for an attribute parameter is a single asterisk "*", which deletes all records in the enrichment datastore.

Value Wildcard

For a fixed attribute, a wildcard search performs over the values of that attribute if the contents (of the value field or **workflowContext**) includes a supported wildcard.

For example, you have set the following:

a. **attribute** "source"

b. **value** "custom_info.lookupkey"

Where "custom_info.lookupkey" contains the string "host2*". In this scenario, the function matches records for the attribute "source" and values "host2", "host201", and "host2a", and deletes all of these records.

Example

The following example demonstrates typical use of Workflow Engine function **deleteEnrichment**.

You want to send an update to your Enrichment API endpoint to delete the data within a field called "source". The field contains a value of "host_1". Set the following:

- **source**: source

The UI translates your settings to the following JSON:

```
{"source":"source"}
```

The request deletes data for the attribute "source" and value "host_1".

It's possible to delete multiple entries for a given attribute by providing a field containing an array of string values in the **source** argument. For example, given the arguments:

```
{"source": "workflowContext.deleteList.source" }
```

Where the **workflowContext** key **deleteList** holds the value:

```
{ "source": [ "node_1", "node_2" ] }
```

In this scenario, the action generates two requests to the API to delete the data for the attribute **source**: one request to delete the value “node_1”, and a second request to delete the value “node_2”. If these requests are successful, the function returns **true** and deletes these values.

deleteTopology

A Workflow Engine function that attempts to delete the named topology. The topology name can be static or a substitution value. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology).

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **deleteTopology** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology)

Example

The following example demonstrates typical use of Workflow Engine function **deleteTopology**.

If you want to delete the topology stored in the "myTopology" key of the workflow context, set the following:

- **topologyName**: workflowContext.myTopology

The UI translates your settings to the following JSON:

```
{"topologyName":"$(workflowContext.myTopology)"}
```

For an alert with the following **workflowContext.myTopology** = "my network", the action deletes the topology.

deleteTopologyLink

A Workflow Engine function that removes a direct link between two endpoints, A (source node) and Z (sink node), in a named topology. Both nodes must exist with a direct link to return true.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **deleteTopologyLink** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology)
sourceNode	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a value, use $(<attribute_name>). For example

			$(custom_info.mySourceNode)
sinkNode	yes	string	The name or substituted value for the 'Z' endpoint (sink node). To substitute a value, use $(<attribute_name>). For example $(custom_info.mySinkNode)

Example

The following example demonstrates typical use of Workflow Engine function **`deleteTopologyLink`**.

If you want to remove a link in the topology "my network" between the alert source and another node you have previously added to the workflow context, set the following:

- **`topologyName`**: my network
- **`sourceNode`**: $(source)
- **`sinkNode`**: $(workflowContext.destination)

The UI translates your settings to the following JSON:

```
{"topologyName":"my network",
"sourceNode":"$(source)","sinkNode":"$(workflowContext.destination)"}
```

For an alert where **`source`** = sflinux101 and the corresponding **`workflowContext.destination`** = sflinux102, the workflow action deletes any existing direct link between the two nodes.

deleteTopologyNode

A Workflow Engine function that deletes a node in the named topology. The node name can be static or a substitution value. To substitute a value, use $(<attribute_name>). For example $(source). As a best practice, set 'first match only' for this action.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`deleteTopologyNode`** takes the following arguments:

Name	Required	Type	Description
topologyName	yes	string	The name or substited value for the topology. To substitute a value, use $(<attribute_name>). For example $(custom_info.myTopology).
nodeName	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a value, use $(<attribute_name>). For example $(custom_info.node).

Example

The following example demonstrates typical use of Workflow Engine function **`deleteTopologyNode`**.

If you want to delete a node in the topology "My Network" for alert source, set the following:

- **`topologyName`**: my network
-)**`sourceNode`**: $(source)

The UI translates your settings to the following JSON:

```
{"topologyName":"my network","nodeName":"$(source)"}
```

For an alert where **source** = sflinux101, the action deletes the node.

deltaEvent

A Workflow Engine function that returns **true**:

- If the specified event fields differ from corresponding fields in an existing alert.
- When an error occurs in the delta check.
- When no alert exists.

Returns **false** when it detects no changes.

Uses shallow object inspection which compensates for list ordering and object key ordering, but may still result in an inaccurate response.

This function has an inherent call to willDeduplicateAlert so you do not need to call it first.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **deltaEvent** takes the following arguments:

Name	Required	Type	Description
fields	Yes	Object	An array of core and custom info fields to check. Objects may produce unexpected results during comparison

Example

The following example demonstrates typical use of Workflow Engine function **deltaEvent**. If you want to check for differences in the services in the custom_info.eventDetails.services field between the alert and deduplicating events, set the following:

- fields: ["custom_info.eventDetails.services"]

The UI translates your settings to the following JSON:

```
["custom_info.eventDetails.services"]
```

Given an alert with the following fields:

```
"custom_info": {"eventDetails":
                    {"services": ["APPSERVER"]}
                  }
```

An event with the following fields returns true:

```
"custom_info": {"eventDetails":
                    {"services": ["NETWORK"]}
                  }
```

An event with the following fields returns false:

```
"custom_info": {"eventDetails":
                    {"services": ["APPSERVER"]}
                  }
```

dnsLookup

A Workflow Engine function that performs a lookup of an IP address or name to return the following object:

```
{
  "name" : <resolved name>,
  "address" : <resolved address>,
  "fqdn" : <resolved fqdn>
}
```

The functon uses the underlying dnsLookup bot utility and caches results. Repeated queries for the same data use the cached values. Caching happens at the Moogfarmd level. You can edit the Moogfarmd configuration if needed. See Java 11 Networking Properties and Java 8 Network Properties for details on caching and configuration.

By default the cache lasts the lifetime of the JVM. You can add the appropriate flags and caching duration to the Moogfarmd startup. The function writes the results from the DNS lookup to the **`workflowContext.dnsDetails`** for use in other actions. Returns null when the DNS lookup fails to find any details for an IP address. To prevent unpredictable behavior, check the returned data for null before using it in a downstream actions.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`dnsLookup`** takes the following arguments:

Name	Required	Type	Description
lookup	yes	string	Object field containing the value to look up.

Example

The following example demonstrates typical use of Workflow Engine function **`dnsLookup`**. To lookup the DNS entry for the source field, set the following:

- lookup: source

The UI translates your settings to the following JSON:

```
{"lookup":"source"}
```

For an alert where **`source`** = 198.51.100.12, the function updates the **`workflowContext`** as follows:

```
"dnsDetails": {
    "address": "198.51.100.12",
    "fqdn": "sflinux101.example.com",
    "name": "sflinux101"
}
```

doesNotHaveStatus

A Workflow Engine function that returns **`true`** when the in-scope alert or Situation is not in any of the specified states.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **doesNotHaveStatus** takes the following arguments:

Name	Required	Type	Description
states	Yes	Object	The states to check for. Choose from: Acknowledged Active Assigned Closed Dormant Opened Resolved SLA Exceeded Unacknowledged Unassigned Not all states are available to both alerts and Situations. For example, you cannot set an alert to Dormant.

Example

The following example demonstrates typical use of Workflow Engine function **doesNotHaveStatus**.

You want to check if an alert is neither Opened, Acknowledged, or Closed before performing subsequent actions in your workflow. Set the following:

- **states** : [“Opened”, “Acknowledged” , “Closed”]
- Forwarding behavior: Stop this workflow. This ensures that if the alert is in any of the specified states, subsequent actions in this workflow do not execute.

The UI translates your settings to the following JSON:

```
states : [ “Owned”, “Acknowledged” , “Closed” ]
```

If the alert is not in any of these states, the function returns **true** and the alert is forwarded to the next action in the workflow.

If the alert is in any of these states, the function returns **false** and the forwarding behaviour prevents subsequent actions in the workflow from executing.

dropEvent

A Workflow Engine function that allows you to prevent further processing of an event. For example, you may wish to discard an event if it has a severity of 0 and willCreateNewAlert returns **true**.

Always returns **`false`**. Follows the Forwarding Behavior settings.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`dropEvent`** has no arguments.

estimateSeverity

A Workflow Engine function that uses a predefined classification algorithm to estimate event or alert severity.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is available for event, alert, and enrichment workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`estimateSeverity`** takes the following arguments:

Name	Required	Type	Description
`eventFields`	No	Object	Array of fields to use in the classification algorithm. Defaults to the **`description`** field.
`severityField`	No	String	Destination field for the classification algorithm's calculated severity. Defaults to the **`severity`** field.

If you do not configure these arguments, the function parses the event description field to calculate a severity value, which it assigns to the severity field.

Example

The following example demonstrates typical use of Workflow Engine function **`estimateSeverity`**.

The optional **`eventFields`** argument allows you to customize the event fields the function uses for severity classification. You define these as an array of event fields. For example, if you set the following:

- **`eventfields`**: ["agent", "description", "custom_info.clustering", "custom_info.enrichment.BusinessApps"]

The UI translates your settings to the following JSON:

```
{ "eventFields": ["agent", "description", "custom_info.clustering",
"custom_info.enrichment.BusinessApps" ]}
```

The optional **`severityField`** argument allows you to assign the estimated severity to a target field instead of using the default, **`severity`**. For example, to assign the result to **`custom_info.catasaurus.severity`**, set the following:

- **`severityField`**: custom_info.catasaurus.severity

The UI translates your settings to the following JSON:

```
{"severityField":"custom_info.catasaurus.severity"}
```

If the classification algorithm fails to estimate the severity and target is the event severity field, the function returns **`false`** and the event does not update. If the target is a custom_info field, the value defaults to Indeterminate.

existingAlertFilter

A Workflow Engine function that returns **`true`** if the existing alert for a deduplicating event matches a SQL-like filter. Uses the **`evaluateFilter`** method in the CEvents API.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`existingAlertFilter`** takes the following arguments:

Name	Required	Type	Description
`filter`	Yes	String	SQL-like filter expression. Use single quotes around strings. For example: **`description matches 'abc'`**.

Example

The following example demonstrates typical use of Workflow Engine function **`existingAlertFilter`**. If you want to verify if a deduplicating event has an agent_location value of "London", set the following:

- filter: agent_location = 'London'

The UI translates your settings to the following JSON:

```
agent_location = 'London'
```

Given an alert with the following signature:

```
"APPSERVER2002:APPLICATION:AVAILABILITY"
```

An event with the following fields returns true:

```
"signature": "APPSERVER2002:APPLICATION:AVAILABILITY",
"agent_location": "London"
```

An event with the following fields returns false:

```
"signature": "APPSERVER2002:APPLICATION:AVAILABILITY",
"agent_location": "SF"
```

exportViaKafka

A Workflow Engine function that exports the payload from a createPayload function to an external Kafka endpoint.

To use this function, you must first configure the following:

- A Kafka Endpoints integration, which configures the endpoints for this function to use.

- A createPayload function which precedes this function, in order to generate the payloads this function exports.
- For best practice, create a new engine to handle the data export process. This is to prevent potential blockages during the export process under load.
- If you want to export both alerts and Situations, you must create a separate engine for each workflow. A separate engine has the following moolet characteristics:

```
standalone_moolet: true
threads: 1
event_handlers: [<if required>]
process_output_of: <place in the moolet chain>
```

Cisco recommends this moolet is single threaded to ensure Cisco Crosswork Situation Manager works at the same rate as the receiving API. However, you can modify the thread count if necessary, for example if the endpoint has inherent rate or load mechanics, or ordering. No other moolet should rely on or process the output of this one.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function. If you use the sweep up filter within the workflow, **`createPayload`** applies to all the objects in the workflow. Consequently, **`exportViaKafka`** exports the payload created using all objects within the workflow.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`exportViaKafka`** takes the following arguments:

Name	Required	Type	Description
`endpointName`	Yes	String	Name of the Kafka endpoint defined in the Kafka Endpoints Reference integration.
`topic`	Yes	String	Destination Kafka topic.
`key`	No	String	Optional Kafka topic key. Allows substitution such as **`'$custom_info.myvalue'`**.

Example

The following example demonstrates typical use of Workflow Engine function **`exportViaKafka`**.

If you want to export to an endpoint with the name Broker1, set the following:

- **`endpointName`**: AlertExport
- **`topic`**: Export Feed
- **`key`**: myKey

The UI translates your settings to the following JSON:

```
{"endpointName":"AlertExport","topic":"myTopic","key":"myKey"}
```

The function returns **`true`** if it was able to locate and successfully complete the export. If it could not find the endpoint configuration, or the export was unsuccessful, the function returns **`false`**.

exportViaRest

A Workflow Engine function that exports the payload from a createPayload to an external REST endpoint.

To use this function, you must first configure the following:

- A REST Endpoints integration, which configures the endpoints for this function to use.
- A createPayload function which precedes this function, in order to generate the payloads this function exports.
- For best practice, create a new engine to handle the data export process. This is to prevent potential blockages during the export process under load.
- If you want to export both alerts and Situations, you must create a separate engine for each workflow. A separate engine has the following Moolet characteristics:

```
standalone_moolet: true
threads: 1
event_handlers: [<if required>]
process_output_of: <place in the moolet chain>
```

Cisco recommends this Moolet is single threaded to ensure Cisco Crosswork Situation Manager works at the same rate as the receiving API. However, you can modify the thread count if necessary, for example if the endpoint has inherent rate or load mechanics, or ordering. No other Moolet should rely on or process the output of this one.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function. If you use the sweep up filter within the workflow, **`createPayload`** applies to all the objects in the workflow. Consequently, **`exportViaRest`** exports the payload created using all objects within the workflow.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`exportViaRest`** takes the following arguments:

Name	Required	Type	Description
`endpointName`	Yes	String	Name of the endpoint defined in the REST Endpoints Reference integration.

Example

The following example demonstrates typical use of Workflow Engine function **`exportViaRest`**.

Set the following:

- **`endpointName`**: AlertExport

The UI translates your settings to the following JSON:

```
{"endpointName":"AlertExport"}
```

The function returns **`true`** if it was able to locate and successfully complete the export. If it could not find the endpoint configuration, or the export was unsuccessful, the function returns **`false`**.

filterByCookbook

A Workflow Engine function that allows you to filter inscape Situations (trigger and swept up) based on their Visualize data. These are inclusive filters and return **`true`** if the Visualize data for the Situation matches the cookbook name.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`filterByCookbook`** takes the following arguments:

Name	Required	Type	Description
cookbook	Yes	String	Name of the cookbook to filter by.

Example

The following example demonstrates typical use of Workflow Engine function **`filterByCookbook`**.

If you want to check if the cookbook name is "Default Cookbook", enter the following:

- **cookbook**: Default Cookbook

The UI translates your settings to the following JSON:

```
{"cookbook":"Default Cookbook"}
```

Given a Situation with the following Visualize data:

```
{
    "visualize": {
        "origin": "Cookbook",
        "cookbook_name": "Default Cookbook",
        "recipe_name": "Description"
    }
}
```

The function returns **`true`**, since the value of **cookbook** matches the Visualize data.

Now consider this filter:

```
{"cookbook":"MyCookbook"}
```

In this case the function returns **`false`**, since the value of **cookbook** does not match the Visualize data.

filterByCookbookAndRecipe

A Workflow Engine function that allows you to filter inscape Situations (trigger and swept up) based on their Visualize data. These are inclusive filters and return **`true`** if the Visualize data for the Situation matches the cookbook name and recipe name.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`filterByCookbookAndRecipe`** takes the following arguments:

Name	Required	Type	Description
`cookbook`	Yes	String	Name of the cookbook to filter by.
`recipe`	Yes	String	Name of the recipe to filter by.

Example

The following example demonstrates typical use of Workflow Engine function **`filterByCookbookAndRecipe`**.

If you want to check if the cookbook name is `"Default Cookbook"` and the recipe name is `"Description"`, enter the following:

- **`cookbook`**: Default Cookbook
- **`recipe`**: Description

The UI translates your settings to the following JSON:

```
{"cookbook":"Default Cookbook","recipe":"Description"}
```

Given a Situation with the following Visualize data:

```
{
    "visualize": {
        "origin": "Cookbook",
        "cookbook_name": "Default Cookbook",
        "recipe_name": "Description"
    }
}
```

The function returns **`true`**, since the values of both **`cookbook`** and **`recipe`** match the Visualize data.

Now consider these filters:

```
{"cookbook":"MyCookbook","recipe":"Description"}
```

In this case the function returns **`false`**, since only the value of **`recipe`** matches the Visualize data, while **`cookbook`** does not. For the function to return **`true`**, the values of both arguments must match the Visualize data.

filterByRecipe

A Workflow Engine function that allows you to filter inscape Situations (trigger and swept up) based on their Visualize data. These are inclusive filters and return **`true`** if the Visualize data for the Situation matches the recipe name.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`filterByRecipe`** takes the following arguments:

Name	Required	Type	Description
`recipe`	Yes	String	Name of the recipe to filter by.

Example

The following example demonstrates typical use of Workflow Engine function **`filterByRecipe`**.

If you want to check if the recipe name is "Description", enter the following:

- **`recipe`**: Description

The UI translates your settings to the following JSON:

```
{"recipe":"Description"}
```

Given a Situation with the following Visualize data:

```
{
    "visualize": {
        "origin": "Cookbook",
        "cookbook_name": "Default Cookbook",
        "recipe_name": "Description"
    }
}
```

The function returns **`true`**, since the value of **`recipe`** matches the Visualize data.

Now consider this filter:

```
{"recipe":"Source"}
```

In this case the function returns **`false`**, since the value of **`recipe`** does not match the Visualize data.

forward

A Workflow Engine function that forwards the object to the named Moolet.
Navigate to Settings> Self Monitoring > Event Processing to see the list of Moolets running on your system.

This function is available for alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`forward`** takes the following arguments:

Name	Required	Type	Description
moolet	Yes	String	Moolet name to forward the object to. Must match the name of a running Moolet.

getEnrichment

A Workflow Engine function that retrieves data from the enrichment data store through the Cisco Crosswork Situation Manager Enrichment API. Returns **`true`** if the request is successful.

This function relates directly to the API details from your Enrichment API.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for alert and enrichment workflows.

This function does not modify the in-scope object when it retrieves enrichment data.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`getEnrichment`** takes the following arguments:

Name	Required	Type	Description
`attribute`	Yes	String	Name of the attribute to lookup. For example, `"hostname"`.
`value`	Yes	String	Name of the field or **`workflowContext`** key holding the data to lookup. To specify a **`workflowContext`** key, prefix with `"workflowContext"`. For example, `"workflowContext.lookupkey"`.
`target`	No	String	Optional subkey under **`custom_info.enrichment`** to assign the result to. For example, `"data"`.

Wildcard Search

You can perform a wildcard search if the event field or **`workflowContext`** contains a wildcard string. Valid wildcards are as follows:

Wildcard Type	Example	Description
Ends with	“*searchString”	Matches strings ending with the string “searchString”.
Begins with	“searchString*”	Matches strings beginning with the string “searchString”.
Contains	“*searchString*”	Matches strings containing the string “searchString”.

The following conditions apply when performing a wildcard search on an attribute or value:

Attribute Wildcard

The only supported wildcard for an attribute parameter is a single asterisk “*”, which searches all records in the enrichment datastore and returns the first.

Value Wildcard

For a fixed attribute, a wildcard search performs over the values of that attribute if the contents (of the value field or **`workflowContext`**) includes a supported wildcard.

For example, you have set the following:

- **`attribute`** “source”
- **`value`** “custom_info.lookupkey”

Where `"custom_info.lookupkey"` contains the string `"host2*"`. In this scenario, the function matches records for the attribute `"source"` and values “host2”, “host201”, and “host2a”, and returns the first record in this list, `"host2"`.

Example

The following example demonstrates typical use of Workflow Engine function **`getEnrichment`**.

Within your Enrichment API endpoint you have an attribute called `"source"`. You want to retrieve data from the **`lookupkey`** field of this attribute and assign the result to a subkey called `"data"`. Set the following:

- **`attribute`**: source
- **`value`**: custom_info.lookupkey
- **`target`**: data

The UI translates your settings to the following JSON:

```
{"attribute":"source","value":"custom_info.lookupkey","target":"data"}
```

The function sends a request to the datastore configured in the Enrichment API endpoint for **`attribute`** `"source"` and the contents of the "custom_info.lookupkey" field as the value to search. If successful, the function returns **`true`**, and any enrichment data in this field returns as JSON, which the function assigns to **`custom_info.enrichment.data`**.

The field or workflowContext key you specify for **`value`** must contain either a string or an array of strings. For an array of strings, the function looks up each string and keys the aggregated results by value.

For example, in the same configuration, now consider that custom_info.lookupkey holds the following array:

```
["host1", "host2"]
```

Assuming the request is successful and returns **`true`**, the function assigns enrichment data to custom_info.enrichment.data in the following format:

```
{
  "host1": { .. enrichment data for host 1 .. },
  "host2": { .. enrichment data for host 2 .. }
}
```

getIntegrationConfig

A Workflow Engine function that retrieves an integration configuration and stores it in the **`workflowContext`** for subsequent actions to use. Returns **`true`** if the specified type and key are found.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`getIntegrationConfig`** takes the following arguments:

Name	Required	Type	Description
`integrationType`	Yes	String	Integration type, for example PayloadMaps.
`integrationKey`	No	String	Key to use within the type. Varies by type, for example map name from the PayloadMaps integration. For integration types with multiple sub-objects, such as maps or endpoints, provide this to

			return only the specific configuration.

Example

The following example demonstrates typical use of Workflow Engine function **`getIntegrationConfig`**.

If the integration you configure is PayloadMaps, set the following:

- **`integrationType`**: PayloadMaps

The UI translates your settings to the following JSON:

```
{"integrationType":"PayloadMaps"}
```

In this example, the PayloadMaps integration has created two payload maps: `"Export"` and `"Datalake"`. The JSON equivalent of this is:

```
"PayloadMaps" : {
  "config" : [{
    [
    {
    "configName": "Export",
        "rules": [{
            "name": "alert_id",
            "rule": "$alert_id",
            "conversion": "stringToInt"
        }, {
            "name": "description",
            "rule": "$description",
            "conversion": "none"
        }, {
            "name": "class",
            "rule": "$class",
            "conversion": "none"
        }, {
            "name": "type",
            "rule": "$type",
            "conversion": "none"
        }, {
            "name": "severity",
            "rule": "$severity",
            "conversion": "unenumerate"
        }, {
            "name": "agent",
            "rule": "$agent:$agent_location",
            "conversion": "none"
        }, {
            "name": "services",
            "rule": "$custom_info.services",
            "conversion": "none"
        }, {
            "name": "servicesList",
            "rule": "$custom_info.services",
            "conversion": "objToString"
        }, {
            "name": "manger",
```

```
            "rule": "Manager:$manager",
            "conversion": "none"
        }
     ]
    },
    {
     "configName": "Datalake",
     "rules": [
         {
             "name": "alert",
             "rule": "$alert_id",
             "conversion": "stringToInt"
         }, {
             "name": "description",
             "rule": "$description",
             "conversion": "none"
         }
     ]
  }
  ]
}
```

As **`integrationKey`** is not set, the function produces a **`workflowContext`** containing both maps:

```
{
    "workflowConfig": {
        "payloadmaps": [
            {
                "configName": "Export",
                "rules": [
                    {
                        "name": "alert_id",
                        "rule": "$alert_id",
                        "conversion": "stringToInt"
                    },
                    {
                        "name": "description",
                        "rule": "$description",
                        "conversion": "none"
                    },
                    {
                        "name": "class",
                        "rule": "$class",
                        "conversion": "none"
                    },
                    {
                        "name": "type",
                        "rule": "$type",
                        "conversion": "none"
                    },
                    {
                        "name": "severity",
                        "rule": "$severity",
                        "conversion": "unenumerate"
                    },
                    {
                        "name": "agent",
                        "rule": "$agent:$agent_location",
                        "conversion": "none"
                    },
                    {
                        "name": "services",
```

```
                        "rule": "$custom_info.services",
                        "conversion": "none"
                    },
                    {
                        "name": "servicesList",
                        "rule": "$custom_info.services",
                        "conversion": "objToString"
                    },
                    {
                        "name": "manger",
                        "rule": "Manager:$manager",
                        "conversion": "none"
                    }
                ]
            },
            {
                "configName": "Datalake",
                "rules": [
                    {
                        "name": "alert",
                        "rule": "$alert_id",
                        "conversion": "stringToInt"
                    },
                    {
                        "name": "description",
                        "rule": "$description",
                        "conversion": "none"
                    }
                ]
            }
        ]
    }
}
```

To only return the export map, set the following:

- **`integrationType`**: PayloadMaps
- **`integrationKey`**: Export

The function now produces a **`workflowContext`** that only contains the Export map:

```
{
  "workflowConfig": {
        "payloadmaps": {
            "configName": "Export",
            "rules": [
                {
                    "name": "alert_id",
                    "rule": "$alert_id",
                    "conversion": "stringToInt"
                },
                {
                    "name": "description",
                    "rule": "$description",
                    "conversion": "none"
                },
                {
```

```
                    "name": "class",
                    "rule": "$class",
                    "conversion": "none"
                },
                {
                    "name": "type",
                    "rule": "$type",
                    "conversion": "none"
                },
                {
                    "name": "severity",
                    "rule": "$severity",
                    "conversion": "unenumerate"
                },
                {
                    "name": "agent",
                    "rule": "$agent:$agent_location",
                    "conversion": "none"
                },
                {
                    "name": "services",
                    "rule": "$custom_info.services",
                    "conversion": "none"
                },
                {
                    "name": "servicesList",
                    "rule": "$custom_info.services",
                    "conversion": "objToString"
                },
                {
                    "name": "manger",
                    "rule": "Manager:$manager",
                    "conversion": "none"
                }
            ]
        }
    }
}
```

getJDBCEnrichment

A JDBC Enrichment Workflow Engine function that adds data to alerts from a JDBC database.

This function relates directly to the database and table definitions from your JDBC Enrichment integration.

This function does not make use of **`workflowContext`**.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is only available for JDBC Enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`getJDBCEnrichment`** takes the following arguments:

Name	Required	Type	Description
`databaseDefName`	Yes	String	Name of the database definition from the JDBC Enrichment integration.

tableDefName	Yes	String	Name of the table definition from the JDBC Enrichment integration
value1	No	String	Alert property to use in the Query field of the table definitions in the JDBC Enrichment integration. For example, "source", or "custom_info.host_name".
value2	No	String	Alert property to use in the Query field of the table definitions in the JDBC Enrichment integration. For example, "source", or "custom_info.host_name".

Example

The following example demonstrates typical use of Workflow Engine function **getJDBCEnrichment**. It assumes you have set up and configured the JDBC Enrichment integration with:

- A database definition of "localmdb".
- A table definition name of "ci"

To retrieve data from a record in an external database and store specific columns in the alert's custom_info, set the following:

- **databaseDefName**: localcmdb
- **tableDefName**: ci

The UI translates your settings to the following JSON:

```
{"databaseDefName":"localcmdb","tableDefName":"ci"}
```

The function retrieves the data and adds it to custom info:

```
{
  "enrichment": {
    "HostDetails": {
      "OS Version": "2.6.9-22.0.1.ELsmp",
      "SupportGroup": "Linux Server",
      "Class": "Linux Server"
    }
  },
  "mooghandling": {
    "isEnriched": true
  }
}
```

getPayload

A Workflow Engine function that creates a **workflowContext** payload from the triggering object from a predefined payload map. For use in subsequent actions, for example exportViaRest or exportViaKafka.

This function relates directly to the payload maps from your Payload Maps integration.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

This function does not modify the in-scope object.

The workflow sweep up filter applies to this function. Swept up objects have an entry under the **`workflowContext.payloads`** object corresponding to their associated ID.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`getPayload`** takes the following arguments:

Name	Required	Type	Description
`mapName`	Yes	String	Name of the map from the Payloads

Example

The following example demonstrates typical use of Workflow Engine function **`getPayload`**.

To create a payload for a map called "AlertExport", set the following:

- **`mapName`**: AlertExport

The UI translates your settings to the following JSON:

```
{ "mapName":"AlertExport"}
```

The function returns **`true`** when it finds a map and creates the payload. It stores the payload in the **`workflowContext`** payloads key. The function also assigns the payload an identifier for the in-scope object.

For example, a payload for an alert with the ID #1234 stores in **`workflowContext`** as follows:

```
"workflowConext" : {
  "payloads" : {
      "1234" : { ... }
  }
}
```

Note

All functions must use the same structure for payloads. For alerts and Situations, use the object ID. For events, use the signature value.

getSituationFlags

A Workflow Engine function that retrieves the Situation flags and stores them in the workflowContext for subsequent actions to use. The function stores the flags in the workflowContext under **`workflowContext.situationFlags`**.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.

Arguments

Workflow Engine function **`getSituationFlags`** has no arguments.

Example

The following example demonstrates typical use of Workflow Engine function **`getSituationFlags`**.

A Situation with the flag "TICKETED" produces the following result:

```
{
    "situationFlags": [
        "TICKETED"
    ]
}
```

getServiceNowEnrichment

A Workflow Engine function that adds data to alerts from a ServiceNow database.

This function does not make use of **`workflowContext`**.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for ServiceNow Enrichment workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`getServiceNowEnrichment`** takes the following arguments:

Name	Required	Type	Description
`tableDefName`	Yes	String	Name of the table definition from the ServiceNow CMDB integration
`value1`	No	String	Alert property to use in the Query field of the table definitions in the ServiceNow CMDB Enrichment integration. For example, `"source"`, or `"custom_info.host_name"`.
`value2`	No	String	Alert property to use in the Query field of the table definitions in the ServiceNow CMDB integration. For example, `"source"`, or `"custom_info.host_name"`.

Example

The following example demonstrates typical use of Workflow Engine function **`getServiceNowEnrichment`**. It assumes you have set up and configured the ServiceNow CMDB integration with:

- A database definition of `"localmdb"`.
- A table definition name of `"ci"`

See Enrich Alerts with ServiceNow data for the full workflow.

To retrieve data from a record in an external database and store specific columns in the alert's custom_info, set the following:

- **`tableDefName`**: ci

The UI translates your settings to the following JSON:

```
{"databaseDefName":"localcmdb","tableDefName":"ci"}
```

The function retrieves the data and adds it to custom info:

```
{
  "enrichment": {
```

```
      "Services": {
        "Client Services": {
          "Apps": "Client Services",
          "SupportGroup": "ITSM Engineering",
          "Class": "Service"
        },
        "Bond Trading": {
          "Apps": "Bond Trading",
          "SupportGroup": "IT Securities",
          "Class": "Service"
        }
      }
    },
    "mooghandling": {
      "isEnriched": true
    }
  }
```

getVisualizationData

A Workflow Engine function that retrieves the Visualize data, including cookbook and recipe details, for a situation. The function stores the details in the workflowContext, under **`workflowContext.visualize`**, which subsequent actions can then utilize.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`getVisualizationData`** has no arguments.

Example

The following example demonstrates typical use of Workflow Engine function **`getVisualizationData`**. Creating a Situation using the "Description" recipe in the "Default Cookbook" produces the following result set:

```
{
    "visualise": {
        "origin": "Cookbook",
        "cookbook_name": "Default Cookbook",
        "recipe_name": "Description"
    }
}
```

hasCausalPRC

A Workflow Engine function that returns **`true`** if one or more alerts in the Situation has a causal PRC flag set.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for Situation workflows only.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **hasCausalPRC** has no arguments.

hasMerged

A Workflow Engine function that returns true if the Situation has been merged with another Situation or superseded by a Situation. See Merge Groups for more information.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **hasMerged** has no arguments.

hasNotMerged

A Workflow Engine function that returns true if the Situation has not been merged with another Situation or superseded by another Situation. See Merge Groups for more information.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **hasNotMerged** has no arguments.

hasSimilarSituations

A Workflow Engine function that returns **true** when the Situation has a similar Situation above the specified threshold. Uses the Situation Similarity utility provided with the Workflow Engine to calculate similarity between .5 (50% similar) and 1 (100% similar).

The Situation Similarity requires some configuration. You can find the configuration file at: **$MOOGSOFT_HOME/config/SimilarSigConfig.conf.**

- Verify the Graze API credentials are valid.
- Verify the webhost is correct if the UI runs on a different host than Moogfarmd.

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **hasSimilarSituations** takes the following arguments:

Name	Required	Type	Description
Similarity	Yes	Number	Similarity threshold between .5 and 1 used to identify similar Situations. Situations with an equal or greater value qualify.

hasStatus

A Workflow Engine function that returns **`true`** when the in-scope alert or Situation is in any of the specified states.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`hasStatus`** takes the following arguments:

Name	Required	Type	Description
`states`	Yes	Object	The states to check for. Choose from: Acknowledged Active Assigned Closed Dormant Opened Resolved SLA Exceeded Unacknowledged Unassigned Not all states are available to both alerts and Situations. For example, you cannot set an alert to Dormant.

Example

The following example demonstrates typical use of Workflow Engine function **`hasStatus`**.

You want to check if an alert is either Opened, Acknowledged, or Closed before performing subsequent actions in your workflow. Set the following:

- **`states`** : [“Opened”, “Acknowledged” , “Closed”]
- Forwarding behavior: Stop this workflow. This ensures that if the alert is not in any of the specified states, subsequent actions in this workflow do not execute.

The UI translates your settings to the following JSON:

```
states : [ “Owned”, “Acknowledged” , “Closed” ]
```

If the alert is in any of these states, the function returns **`true`** and the alert is forwarded to the next action in the workflow.

If the alert is not in any of these states, the function returns **`false`** and the forwarding behaviour prevents subsequent actions in the workflow from executing.

isAlertAcknowledged

A Workflow Engine function that returns **`true`** when the in-scope alert state is Acknowledged.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for alert and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`isAlertAcknowledged`** has no arguments.

Example

You can use this function as a qualifier or condition, so that subsequent actions only execute if the alert state is Acknowledged.

isAlertNotAcknowledged

A Workflow Engine function that returns **`true`** when the in-scope alert state is not Acknowledged.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for alert and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`isAlertNotAcknowledged`** has no arguments.

Example

You can use this function as a qualifier or condition, so that subsequent actions only execute if the alert state is not Acknowledged.

isAssigned

A Workflow Engine function that returns true if the object has an owner or moderator.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`isAssigned`** has no arguments.

isClear

A Workflow Engine function that returns **`true`** if the object's severity level is Clear (0).

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`isClear`** has no arguments.

isInSubnet

A Workflow Engine function that returns true when an IP address is present within a specified subnet.

This function is available for event, alert, and enrichment workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`isInSubnet`** takes the following arguments:

Name	Required	Type	Description
field	Yes	String	Event or alert field to retrieve the IP address from.
subnet	Yes	String	Subnet to check for membership, expressed as: Classless Inter-Domain Routing (CIDR): **`nnn.nnn.nnn.nnn/nn`**. For example: 198.51.100.0/24 Mask: **`nnn.nnn.nnn.nnn/nnn.nnn.nnn.nnn`**. For example: 198.51.100.0/255.255.255.0

Example

The following example demonstrates typical use of Workflow Engine function **`isInSubnet`**. To check if the source field for an IP address within the `"198.51.100.0/24"` subnet, set the following:

- field: source_id
- subnet: 198.51.100.0/24

The UI translates your settings to the following JSON:

```
{"field":"source_id","subnet":"(198.51.100.0/24)"}
```

An object with the following **`source_id`** value returns **`true`**:

```
source_id":"198.51.100.33"
```

An object with the following **`source_id`** value returns **`false`**:

```
source_id":"198.51.101.33"
```

isNewerThan

A Workflow Engine function that returns true when the object age in seconds is less than a specified age in seconds. Uses the difference between the current time and **`agent_time`** for events, **`int_last_event_time`** for alerts, and **`last_event_time`** for Situations.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **isNewerThan** takes the following arguments:

Name	Required	Type	Description
age	Yes	Number	Maximum object age in seconds.

isNotAssigned

A Workflow Engine function that returns true if the object does not have an owner or moderator.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **isNotAssigned** has no arguments.

isNotClear

A Workflow Engine function that returns **true** if the object's severity level is not "Clear".

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **isNotClear** has no arguments.

isNotNull

A Workflow Engine function that returns **true** if the value for a field within an object is not null, is not an empty object , **{}**, or is not an empty array , **[]**. See Alert and Event Field Reference and Event and Alert Field Best Practice for more information on object fields.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **isNotNull** takes the following arguments:

Name	Required	Type	Description
field	Yes	String	Object field.

Example

The following example demonstrates typical use of Workflow Engine function **isNotNull**.

```
{"field":"custom_info"}
```

isNull

A Workflow Engine function that returns **`true`** if the value for a field within an object is null, is an empty object , **`{}`**, or is an empty array , **`[]`**. See Alert and Event Field Reference and Event and Alert Field Best Practice for more information on object fields.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`isNull`** takes the following arguments:

Name	Required	Type	Description
`field`	Yes	String	Object field.

Example

The following example demonstrates typical use of Workflow Engine function **`isNull`**.

```
{"field":"custom_info"}
```

isOlderThan

A Workflow Engine function that returns true when the object age in seconds is greater than a specified age in seconds. Uses the difference between the current time and **`agent_time`** for events, **`int_last_event_time`** for alerts, and **`last_event_time`** for Situations.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`isOlderThan`** takes the following arguments:

Name	Required	Type	Description
age	Yes	Number	Minimum object age in seconds.

labelSituation

A Workflow Engine function that labels the Situation using the Situation Manager Labeler macro language. Does not override manual Situation descriptions. See Situation Manager Labeler for more information on the macro language.

This function is available for Situation workflows only.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`labelSituation`** takes the following arguments:

Name	Required	Type	Description

label	Yes	String	Macro-based label to use for the Situation. Standard macros can be used.

listContains

A Workflow Engine function that returns true when the array field you query contains some of your specified values. Define values as an array, for example [a] or [a, b, c]. You must specify an array field, not a string.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`listContains`** takes the following arguments:

Name	Required	Type	Description
field	Yes	String	Name of the array field to check values in. You can specify custom info fields. The specified field must be an array of values not a string.
values	Yes	Object	List of values to check for, any intersection is valid. Define values as an array, for example [a] or [a, b, c].

Example

The following example demonstrates typical use of Workflow Engine function **`listContains`**. If you want to check for the existence of "Webserver" or "Webapp" elements in an array of services in **`custom_info`**, set the following:

- field: custom_info.eventDetails.service_list
- values: ["Webserver","Webapp"]

The UI translates your settings to the following JSON:

```
{"field":"custom_info.eventDetails.service_list","values":["Webserver","Webapp"]
}
```

An object with the following **`custom_info`** value returns **`true`**:

```
"custom_info": {"eventDetails":
                    {"service_list": ["Webapp"] }
                }
```

An object with the following **`custom_info`** value returns **`false`**:

```
"custom_info": {"eventDetails":
                    {"service_list": ["Network"] }
                }
```

listContainsAll

A Workflow Engine function that returns true when the array field you query contains all of your specified values. Define values as an array, for example [a] or [a, b, c]. You must specify an array field, not a string.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`listContainsAll`** takes the following arguments:

Name	Required	Type	Description
field	Yes	String	Name of the array field to check values in. You can specify custom info fields. The specified field must be an array of values not a string.
values	Yes	Object	List of values to check for, any intersection is valid. Define values as an array, for example [a] or [a, b, c].

Example

The following example demonstrates typical use of Workflow Engine function **`listContainsAll`**. If you want to check for the existence of both `"Webapp"` and `"Webserver"` elements in an array of services in **`custom_info`**, set the following:

- field: custominfo.eventDetails.service_list
- values: `["Webapp","Webserver"]`

The UI translates your settings to the following JSON:

```
{"field":"custominfo.eventDetails.service_list","values":["Webapp","Webserver"]}
```

An object with the following **`custom_info`** value returns **`true`**:

```
"custom_info": {"eventDetails":
                    {"service_list": [ "Webserver", "Webapp"] }
                }
```

An object with the following **`custom_info`** value returns **`false`**:

```
"custom_info": {"eventDetails":
                    {"service_list": ["Webserver","Network"]}
                }
```

listDoesNotContain

A Workflow Engine function that returns true when the array field you query contains none of your specified values. Define values as an array, for example [a] or [a, b, c]. You must specify an array field, not a string.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`listDoesNotContain`** takes the following arguments:

Name	Required	Type	Description
field	Yes	String	Name of the array field to check values in. You can specify custom info fields.

			The specified field must be an array of values not a string.
values	Yes	Object	List of values to check for, any intersection returns false. Define values as an array, for example [a] or [a, b, c].

Example

The following example demonstrates typical use of Workflow Engine function **`listDoesNotContain`**. If you want to check for the absence of an element "Network" in an array of services in **`custom_info`**, set the following:

- field: custominfo.eventDetails.service_list
- values: ["Network"]

The UI translates your settings to the following JSON:

```
{"field":"custominfo.eventDetails.service_list","values":["Network"]}
```

An object with the following **`custom_info`** value returns **`true`**:

```
"custom_info": {"eventDetails":
                    {"service_list": ["Webapp","Webserver"] }
                }
```

An object with the following **`custom_info`** value returns **`false`**:

```
"custom_info": {"eventDetails":
                    {"service_list": ["Webapp","Webserver","Network"] }
                }
```

logCEvent

A Workflow Engine function that prints a warning level message containing the current in-scope object in a readable JSON format to the Moogfarmd log file.

You can use this function to debug workflows. It is not recommended for production workflow because it can clutter log files and make them difficult to use.

This function is available for event, alert, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`logCEvent`** takes the following arguments:

Name	Required	Type	Description
tag	no	string	Optional text to print with the log message to help you find related messages.

Example

The following example demonstrates typical use of Workflow Engine function **`logCEvent`**.

Set the following:

- **tag**: workflowdebug

The UI translates your settings to the following JSON:

```
{"tag":"workflowdebug"}
```

logMessage

A Workflow Engine function that logs a warning level message to the Moogfarmd log. See Configure Logging for information on log locations.

Prepends the message with the workflow function name and the alert or Situation ID, or signature for events.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **logMessage** takes the following arguments:

Name	Required	Type	Description
message	Yes	String	The message to log.

Example

The following example demonstrates typical use of Workflow Engine function **logMessage**.

```
{"message":"Urgent action required."}
```

logWorkflowContext

A Workflow Engine function that logs the contents of **workflowContext** to the current Moogfarmd log file at a warning level.

This function is for debugging and troubleshooting purposes. Do not use it in a regular production workflow.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **logWorkflowContext** has no arguments.

Example

The following example demonstrates typical use of Workflow Engine function **logWorkflowContext**.

For identification purposes, each **workflowContext** log file entry provides the alert or Situation ID, or the event signature:

```
WORKFLOW CONTEXT: ALERT: 23 :  :
{
    "payloads": [
        {
            "alert_id": 23,
            "description": "Ping fail 10.0.0.1",
            "class": "wqtooling",
```

```
                "type": "RestTest",
                "severity": "Minor",
                "agent": "RESTLAM:rest_test.js",
                "services": [
                    "a",
                    "b",
                    "c"
                ],
                "servicesList": "[\"a\",\"b\",\"c\"]",
                "manger": "Manager:RESTLam1"
            }
        ],
        "location": {
            "city": {
                "mycity": {
                    "mytown": "London"
                }
            }
        }
    }
```

logWorkflowDuration

A Workflow Engine function that logs debug messages for the workflow execution duration. To log duration you need to set at lest two actions with logWorkflowDuration in your workflow. The first starts the timer. subsequent instances log the elapsed time since the first logWorkflowDuration action within the workflow. For example, for an event:

```
DEBUG: [0:Event Workflows housekeeping][20191002 21:08:30.208 -0400]
[WorkflowEngine.js:6907] +|Even
t Workflows::logWorkflowDuration: Workflow  for event :
APPSERVER2002:APPLICATION:AVAILABILITY:  execution
 time = 10858ms|+
```

To enable debug logging for Moogfarmd, execute the following:

```
farmd_cntl --loglevel debug
```

When you are through logging, reset the log level to warn:

```
farmd_cntl --loglevel warn
```

This function is available as a feature of the Workflow Engine v1.0 and later.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`auditWorkflow`** takes the following arguments:

Name	Required	Type	Description
`workflowName`	Yes	String	Optional workflow name which makes it easier to find messages in the log.

lookupAndReplace

Updates a specified alert field with a static value if any listed alert fields contain a text substring or match against a regular expression.

For example, you can search both the **`class`** and **`description`** fields for the words `"router"` or `"switch"` while also searching for the regular expression representing a network interface: `"eth\\d+"`. In case of a match, you can update the **`custom_info.key`** field to the static value of `"network."` Then you can configure a Cookbook recipe to use the **`custom_info.key`** field for clustering.

This function is available for alert and enrichment workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`lookupAndReplace`** takes the following arguments:

Name	Required	Type	Description
`wordList`	Yes	Object	An array of words to look for.
`reList`	Yes	Object	An array of regular expressions to test for. Use JavaScript notation for regular expressions.
`inFields`	Yes	Object	An array of alert fields to check. Allows **`custom_info`** fields.
`alertField`	Yes	String	Alert field to update if one of the inFields contains a word from the **`wordList`** or matches a regular expression from the **`reList`**.
`value`	Yes	String	Static value to set for the **`alertField`**.

Example

The following example demonstrates typical use of Workflow Engine function **`lookupAndReplace`**. Set the following to search for network related terms in the **`class`** or **`description`** fields and set the **`custom_info.key`** field to `"network"`:

- **`wordList`**: ["router","switch"]
- **`reList`**: ["eth\\d+","network"]
- **`inFields`**: ["class","description"]
- **`alertField`**: custom_info.services
- **`value`**: network

The UI translates your settings to the following JSON:

```
{"wordList":["router","switch"],"reList":["eth\\d+","network"],"inFields":["clas
s","description"],"alertField":"custom_info.key","key":"network"}
```

The following example data matches the lookup criteria:

```
"class":"network",
"description":"Communication link failure."

 "class":"",
 "description":"Router failed."
```

```
"class":"",
"description":"Interface eth0 down."
```

For all matching cases, the Workflow Engine updates the **custom_info** field as follows:

```
"custom_info": {"key": "network"}
```

The following data does not match the lookup criteria so the **custom_info.key** field remains unchanged:

```
"class":"",
"description":"Error establishing database connection."
```

lowerCase

A Workflow Engine function that changes the value of a field to lower case. For example, changes a value of "NETWORK" to "network".

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **lowerCase** takes the following arguments:

Name	Required	Type	Description
field	Yes	String	Name of the field.

The following example demonstrates typical use of Workflow Engine function **lowerCase**.

To change the value of the source field of an event to lower case, set the following:

- **field**: source

The UI translates your settings to the following JSON:

```
{"field":"source"}
```

populateNamedTopology

A Workflow Engine function that populates the named topology field **custom_info.moog_topology** with a value. It can be a string value or the value of an alert attribute.

Before populating the field the function checks that the argument value is a valid topology.

This function is available for event and alert workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`populateNamedTopology`** takes the following arguments:

Name	Required	Type	Description
`topologyName`	Yes	String	Topology name or alert attribute name. Use **`$`** for an alert attribute. See the example for more information.

prependFields

A Workflow Engine function that prepends a concatenated set of fields to an existing field, using a separator character.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`prepEndFields`** takes the following arguments:

Name	Required	Type	Description
`sourceFields`	Yes	Object	An array of fields to concatenate, in the format: **`[ field1, ..., fieldn ]`**.
`separator`	Yes	String	Separator to use between the concatenated values. Do not use quotes around the separator.
`destination`	Yes	String	Field to prepend the concatenated string to.

prependString

A Workflow Engine function that prepends a string to an existing field, using a separator character.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`prependString`** takes the following arguments:

Name	Required	Type	Description
`string`	Yes	String	String to prepend.
`destination`	Yes	String	Field to prepend the concatenated string to.

Example

The following example demonstrates typical use of Workflow Engine function **`prependString`**.

```
{"string":"This is an example of prepending further
description.","destination":"description"}
```

removeSituationFlag

A Workflow Engine function that removes a specific flag from a Situation.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`removeSituationFlag`** takes the following arguments:

Name	Required	Type	Description
`flag`	Yes	String	Flag to remove.

Example

The following example demonstrates typical use of Workflow Engine function **`removeSituationFlag`**. If you want to remove the "TICKET_PENDING" flag from a Situation, enter the following:

- **`flag`**: TICKET_PENDING

The UI translates your settings to the following JSON:

```
{"flag":"TICKET_PENDING"}
```

Given a Situation with the following flag:

```
{
    "situationFlags": [
        "TICKET_PENDING"
    ]
}
```

The Workflow Engine updates the object as follows:

```
{
    "situationFlags": []
}
```

replaceString

A Workflow Engine function that replaces a string or regular expression in a field with a specified string.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`replaceString`** takes the following arguments:

Name	Required	Type	Description
`field`	Yes	String	Field to replace text in.
`replace`	Yes	String	Original string to replace. This value is treated as a regex. Do not include leading or trailing delimiters.
`with`	No	String	New string you want to use instead. If you leave this field blank, replaces the original string with a blank space.

Example

The following example demonstrates typical use of Workflow Engine function **`replaceString`**.

Some systems abbreviate "database" as "d.b." or "D.B.". If you had a **class** field that contains the value "A D.B. has failed", and you wanted to replace the abbreviation with "database", set the following:

- **field**: class
- **replace**: d\.b\.
- **with**: database

The UI translates your settings to the following JSON:

```
{"field":"class","replace":"d\\.b\\.","with":"database"}
```

The function replaces any occurrences of "d.b." and "D.B.", so the resulting value reads "A database has failed".

resolveNotification

A Workflow Engine function that automatically resolves a notification for a service.

This function currently supports the PagerDuty, OpsGenie and xmatters integrations.

This function is available as a feature of 7.4 integrations.

This function requires you to have already configured the services you want to use it with. When you configure some integrations, Cisco Crosswork Situation Manager automatically creates a workflow with the createNotification function; ensure that this workflow is active before you configure the **resolveNotification** function. Integrations this function applies to indicate their compatibility on the UI.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **ackNotification** takes the following arguments:

Name	Required	Type	Description
services	Yes	String	Comma separated list of the service names.

Example

The following example demonstrates typical use of Workflow Engine function **resolveNotification**.

After you have configured the PagerDuty integration, you can configure a workflow with this function to automatically resolve alerts or Situations that Cisco Crosswork Situation Manager sends to PagerDuty.

- **services**: PagerDuty

The UI translates this setting to the following JSON:

```
{"services":"PagerDuty"}
```

Now when Cisco Crosswork Situation Manager sends alert or Situation data to PagerDuty, the corresponding PagerDuty incident is automatically set to "Resolved".

resolveSituation

A Workflow Engine function that marks in-scope Situations as Resolved if they match the workflow's entry filter and sweep up filter. Adds a resolving thread to the Situation that indicates this function resolved it.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is available for Situation workflows only.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`resolveSituation`** has no arguments.

The following example demonstrates typical use of Workflow Engine function **`resolveSituation`**.

Set the following:

The UI translates your settings to the following JSON:

restAsyncPost

A Workflow Engine function that makes a HTTP POST request with a JSON payload to a named REST endpoint. Expects the payload from a previous action, for example, from the convertToJSON function that converts an event, alert or Situation to a JSON blob. Returns **`false`** when no payload is found.

restAsyncPost is a non-blocking asynchronous call which returns **`true`** to the workflow immediately. It is best for a 'data sink' use case. It does not support setting authentication or other HTTP request fields or attributes.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`restAsyncPost`** takes the following arguments:

Name	Required	Type	Description
URL	Yes	String	The URL of the REST endpoint.

The following example demonstrates typical use of Workflow Engine function **`restAsyncPost`**.

```
{"URL":"https://example.com"}
```

reviveSituation

A Workflow Engine function that revives (sets to Open) a Situation that is currently set to Resolved. Provides a way to revive a closed Situation if a support ticket relating to it is still active.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`reviveSituation`** has no arguments.

The following example demonstrates typical use of Workflow Engine function **`workflowEngineFunction`**.

Set the following:

The UI translates your settings to the following JSON:

searchAndReplace

A Workflow Engine function that matches a regular expression to an object field and updates the values for fields in the object based upon a map. You can map the contents of subgroups to other fields. For example, extract the 'source' value inside a **`description`** and map it to the **`source`** field. You can also map fields to a constant value.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`searchAndReplace`** takes the following arguments:

Name	Required	Type	Description
`field`	Yes	String	Field to search.
`expression`	Yes	String	Regular expression pattern to use on the field.
`map`	Yes	Object	Map to apply the extracted values to as a key: value pairing using **`$extract.n`**, where **`n`** = the subgroup identified. For example: **`{ "custom_info.newValue" : "$extract.1", "source" : "$extract.2" }`**

Note

The code display for the Workflow Engine double-escapes characters. You do not need to double-escape in the data entry field. For example the IP address: "((?:\d+\.){3}\d+)".

When you have nested subgroups, as in the example with the IP address, they do not affect the extract numbering.

Example

The following example demonstrates typical use of Workflow Engine function **searchAndReplace**. You can check for an IP address, and a value of "memory" or "disk" in the object's **description** field. When the Workflow Engine finds a match, it maps the following fields:

- source to the matching IP address: **((?:\\d+\\.){3}\\d+)**.
- class to the matching value of "memory" or "disk": **(memory|disk)**.
- custom_info.support team to the constant "NOC".

Set the following:

- **field**: description
- **expression**: ^.+?((?:\d+\.){3}\d+).+?(memory|disk).+?$
- **map**: {"source":"$extract.1","class":"$extract.2","custom_info.support_team":"NOC"}

The UI translates your settings to the following JSON:

```
{"field":"description",
"expression":"^.+?((?:\\d+\\.){3}\\d+).+?(memory|disk).+?$",
"map":{"source":"$extract.1","class":"$extract.2","custom_info.support_team":"NO
C"}}
```

An object with the following description matches the regular expression test:

```
"description": "Host 198.51.100.0 high memory utilization on
mytestbox.example.com"
```

The Workflow Engine updates the object fields as follows:

```
"source": "198.51.100.0",
"custom_info": {"support_team": "NOC"},
"class": "memory"
```

searchAndReplaceOrdered

A Workflow Engine function that matches a regular expression to an object field and updates the values for fields in the object based upon a map. You can map the contents of subgroups to other fields. For example, extract the 'source' value inside a **description** and map it to the **source** field. You can also map fields to a constant value.

searchAndReplaceOrdered requires you to, with the exception of the **$extract.n** pattern, delimit field replacements with "**$[<field>]**". For example, **$[description]**. Otherwise, this function treats the replacement as a literal string.

This function differs from searchAndReplace in that you can provide the map as an array to preserve the mapping order. For efficiency reasons, only use this function instead if you require this functionality, or intend to supply the map as a set of key:value pairs.

For example, the ordered map:

```
[
    {"source": "${source]-1"},
```

```
    {"description": "$[description] $[source]"}
]
```

differs from the unordered map:

```
{
    "source": "${source]-1",
    "description": "$[description] $[source]"
}
```

This is because, given an event with **source** set to "host" and **description** set to "Failure for", the ordered map results in an updated event with **source: "host-1"** and **description: "Failure for host-1"**. The unordered version has the same source, but the description is only "Failure for host", as it doesn't have access to the updated source value from the first operation.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **searchAndReplaceOrdered** takes the following arguments:

Name	Required	Type	Description
field	Yes	String	Field to search.
expression	Yes	String	Regular expression pattern test against the field.
map	Yes	Object	Map to apply the extracted values to as a key : value pairing using **$extract.n**, where **n** = the subgroup identified. For example **[{ "custom_info.newValue" : "$extract.1" }, { "source" : "$extract.2" }, {"description" : "$[description] $extract.3" }]**.

Note

The code display for the Workflow Engine double-escapes characters. You do not need to double-escape in the data entry field. For example the IP address: "((?:\d+\.){3}\d+)".

When you have nested subgroups, as in the example with the IP address, they do not affect the extract numbering.

Example 1

The following example demonstrates typical use of Workflow Engine function **searchAndReplaceOrdered**.

Set the following:

- **field**: description
- **expression**: Event for (host\d+)
- **map**:
  [{"custom_info.eventDetails.manager":"$[source]"},{"source":"$extract.1"},{"description":"$[class] $[type] event: destination $[source] unreachable"}]

This defines the following mapping:

- Save the original value of **source** as the value of **manager**.
- Replace the original value of **source** with an extract from **description**.
- Update the **description** with a statement which references the values of **class**, **type**, and the updated **source** field.

The UI translates your settings to the following JSON:

```
{
  "field": "description",
  "expression": "Event for (host\d+)",
  "map": [
    {
      "manager": "$[manager]::$[source]"
    },
    {
      "source": "$extract.1"
    },
    {
      "description": "$[class] $[type] event: destination $[source] unreachable"
    }
  ]
}
```

With this mapping, given the following event:

```
{
  "signature": "network::availability::host10",
  "source_id": "192.168.1.1",
  "manager": "Pinger",
  "source": "ping-host1",
  "class": "network",
  "agent": "RESTLam",
  "type": "availability",
  "severity": 5,
  "description": "Event for host10",
  "agent_time": 1581951814000,
  "custom_info": = {}
}
```

The function transforms the event payload to:

```
{
  "signature": "network::availability::host10",
  "source_id": "192.168.1.1",
  "manager": "Pinger::ping-host-1",
  "source": "host10",
  "class": "network",
  "agent": "RESTLam",
  "type": "availability",
  "severity": 5,
  "description": "network availability event: destination host10 unreachable",
  "agent_time": 1581951814000,
  "custom_info": = {}
}
```

Example 2

This example makes use of the mapping order to update the description using a source value that a previous mapping assigned.

You can provide the map as an array to preserve the mapping order. For efficiency reasons, only use this functionality if you require it. Otherwise, supply the map as a set of key:value pairs. For example:

map: **{"custom_info.eventDetails.manager":"$[source]" , "source":"$extract.1", "description":"$[class] $[type] event: destination $[source] unreachable"}**

This defines the following mapping:

- **map**: **{"custom_info.eventDetails.manager":"$[source]" , "source":"$extract.1", "description":"$[class] $[type] event: destination $[source] unreachable"}**
- This defines the following mapping:
    - Save the original value of **source** as the value of **manager**.
    - Replace the original value of **source** with an extract from **description**.
    - Update the **description** with a statement which references the values of **class**, **type**, and the original **source** field.

With the same **field** and **expression** arguments as Example 1, the UI translates your settings to the following JSON:

```
{
  "field": "description",
  "expression": "Event for (host\d+)",
  "map": {
      "manager": "$[manager]::$[source]",
      "source": "$extract.1",
      "description": "$[class] $[type] event: desination $[source] unreachable"
  }
}
```

With this mapping, given the same event as before:

```
{
  "signature": "network::availability::host10",
  "source_id": "192.168.1.1",
  "manager": "Pinger",
  "source": "ping-host1",
  "class": "network",
  "agent": "RESTLam",
  "type": "availability",
  "severity": 5,
  "description": "Event for host10",
  "agent_time": 1581951814000,
  "custom_info": = {}
}
```

The event payload is now:

```
{
  "signature": "network::availability::host10",
  "source_id": "192.168.1.1",
  "manager": "Pinger::ping-host-1",
  "source": "host10",
  "class": "network",
  "agent": "RESTLam",
  "type": "availability",
```

```
  "severity": 5,
  "description": "network availability event: destination ping-host-1
unreachable",
  "agent_time": 1581951814000,
  "custom_info": = {}
}
```

description now contains the original value of **source** as this time you have defined **map** as key:value pairs rather than an array.

sendMooletInform

A Workflow Engine function that sends a Moolet inform with a subject and details. Adds the object to the payload, and so is always available to the receiver. See Moolet Informs for more information.

This function is available for alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **sendMooletInform** takes the following arguments:

Name	Required	Type	Description
target	Yes	String	Moolet to send the inform to.
subject	No	String	Subject of the inform.
details	Yes	Object	A JSON object with the details for the inform.

sendToAnsible

A Workflow Engine function that sends an automation request to Ansible.

This function relates directly to configurations from your Ansible Automation integration.

sendToAnsible requires a setAnsibleJob function that precedes it in your workflow.

This function is typically the last action in a workflow. After this action completes you can forward your alert or Situation data to another workflow for further processing. For example, if you want to send alert data to another automation tool.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is only available for automation alert and automation Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **sendToAnsible** has no arguments.

Example

The following example demonstrates typical use of Workflow Engine function **sendToAnsible**, in which you send Ansible a request to restart a service when Cisco Crosswork Situation Manager receives a new alert. It assumes you have set up and configured the following:

- An Ansible Automation integration with the name "Ansible1"
- A setAnsibleJob function where you have defined the automation solution, instance, and payload. You must configure these before you invoke the **sendToAnsible** function in your workflow.
- You have configured the setAnsibleJob arguments as follows:
  - **instance**: Ansible1
  - **jobTemplateName**: Restart-service

The Restart-service template you specify in the Ansible Automation integration defines mapping rules which build the request payload. In this scenario, one of the rules sets the **extra_vars.serviceName** field in the request to the alert's **$source_id**, so the Ansible job tries to restart the service using this value.

Note

You can use **extra_var** settings to pass additional information to Ansible job templates for the associated Ansible playbook to use.

Set the following:

- Forwarding Behavior: Always Forward.

If the request is successful, the function sets the alert or Situation's custom info status field to Pending. Otherwise, it sets to Failed. Automation results from the Ansible automation tool send back through a webhook that uses the Cisco Crosswork Situation Manager integration gateway generic endpoint. See Ansible Automation.

sendToAutomation

A Workflow Engine function that sends an automation request.

This function currently supports the eyeShare and Ignio integrations and directly relates to configurations from these integrations.

sendToAutomation requires a setAutomationPayload function that precedes it in your workflow.

This function is typically the last action in your workflow. After this action completes you can forward your alert or Situation data to another workflow. For example, if you want to send alert data to another automation tool.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is only available for automation alert and automation Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **sendToAutomation** has no arguments.

Example

The following example demonstrates typical use of Workflow Engine function **sendToAutomation**. It assumes you have set up and configured the following:

- An eyeShare integration with the name "eyeShare1"

- A setAutomationPayload function where you have defined the automation solution, instance, and payload. You must configure these before you invoke **sendToAutomation** in your workflow.

Set the following:

- Forwarding Behavior: Always Forward.

If the request is successful, the function sets the alert or Situation's custom info status field to Pending. Otherwise, it sets to Failed. Automation results from the automation tool send back through either a Moolet Informs module or direct update to custom_info. See eyeShare and Ignio for more information.

sendToPuppet

A Workflow Engine function that sends an automation request to Puppet.

This function relates directly to configurations from your Puppet integration.

sendToPuppet requires a setPuppetAutomation function that precedes it in your workflow.

This function is typically the last action in your workflow. After this action completes you can forward your alert or Situation data to another workflow. For example, if you want to send alert data to another automation tool.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is only available for automation alert and automation Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **sendToPuppet** has no arguments.

Example

The following example demonstrates typical use of Workflow Engine function **sendToPuppet**. It assumes you have set up and configured the following:

- A Puppet integration with the name "Puppet1".
- A setPuppetAutomation function where you have defined the automation solution, instance, and payload. You must configure these before you invoke **sendToPuppet** in your workflow.

Set the following:

- Forwarding Behavior: Always Forward.

If the request is successful, the function sets the alert or Situation's custom info status field to Pending. Otherwise, it sets to Failed. Automation results from the Puppet automation tool send back through either a Moolet Informs module or direct update to custom info. See Puppet for more information.

sendToWorkflow

A Workflow Engine function that sends the in-scope object to a named workflow in an informs based engine. This allows for additional flexibility in workflow execution. You can step out of the currently executing workflow while avoiding some of the complexities of the skip function. The function is a wrapper for the MoogDBv2 sendToWorkflow API.

The destination Workflow Engine must be an informs based engine. Informs engines execute only the named workflow without executing subsequent workflows within the engine. By default you can choose the Alert Inform Engine or the Situation Inform Engine.

The **sendToWorkflow** function does note stop the current workflow after it executes. If you want to stop subsequent workflows, use the **stop** function after **sendToWorkflow**.

This function is available for event, alert, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Name	Required	Type	Description
engineName	yes	string	Name of an inform based workflow engine. For example, Alert Inform Engine or Situation Inform Engine.
workflowName	yes	string	Name of the workflow within the inform based engine.
context	no	object	Optional workflow context object. For example {"key":"value"}. If you don't supply a context, sends the currently active context.

Example

The following example demonstrates typical use of Workflow Engine function **sendToWorkflow**. Imagine you want to send an alert to a workflow named "Export Data" in the "Alert Inform Engine" that performs an asynchronous alert data export outside the linear alert workflow. The workflows in Alert Workflows are not blocked waiting for the export to complete.

Set the following:

- **engineName**: Alert Inform Engine
- **workflowName**: Export Data
- **context**: {"details":"Export example: $(alert_id)"}

The UI translates your settings to the following JSON:

```
{"engineName":"Alert Inform Engine","workflowName":"Export
Data","context":{"details":"Export example: $(alert_id)"}}
```

For an alert with id 28, the Workflow Engine passes the alert to the Export Data workflow in the Alert Inform Engine with the following context:

```
{"details":"Export example: 28"}
```

You can use the **createPayload** and **copyToPayload** actions in the Export Data workflow to prepare the alert data for export. **copyToPayload** has access to the workflow context you sent. To add the workflow context data to the export, set the following:

- **payloadKey**: details
- **value**: $(workflowContext.details)

Finally, set up an export action to export the data.

sendViaRest

A Workflow Engine function that sends the payload from a createPayload function to an external REST endpoint.

This function is available as a feature of the Add-ons v1.4 download and later.

To use this function, you must first configure the following:

- A REST Endpoints integration, which configures the endpoints for this function to use.
- A createPayload function which precedes this function, in order to generate the payloads this function sends.
- For best practice, create a new engine to handle the send process. This is to prevent potential blockages during the send process under load.If you want to send both alerts and Situations, you must create a separate engine for each workflow. A separate engine has the following moolet characteristics:

```
standalone_moolet: true
threads: 1
event_handlers: [<if required>]
process_output_of: <place in the moolet chain>
```

Cisco recommends this moolet is single threaded to ensure Cisco Crosswork Situation Manager works at the same rate as the receiving API. However, you can modify the thread count if necessary, for example if the endpoint has inherent rate or load mechanics, or ordering. No other moolet should rely on or process the output of this one.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function. If you use the sweep up filter within the workflow, **`createPayload`** applies to all the objects in the workflow. Consequently, **`sendViaRest`** sends the payload created using all objects within the workflow.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`sendViaKafka`** takes the following arguments:

Name	Required	Type	Description
`endpointName`	Yes	String	Name of the endpoint defined in the REST Endpoints integration.

Example

The following example demonstrates typical use of Workflow Engine function **`sendViaRest`**.

Set the following:

- **`endpointName`**: AlertToSend

The UI translates your settings to the following JSON:

```
{"endpointName":"AlertToSend"}
```

The function returns **`true`** if it was able to locate and successfully send the alert data to the REST endpoint. If it could not find the endpoint configuration, or send the data, the function returns **`false`**.

setAgent

A Workflow Engine function that sets the Agent field of the alert.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setAgent`** takes the following arguments:

Name	Required	Type	Description
`agent`	Yes	String	Agent value to set.

setAgentLocation

A Workflow Engine function that sets the Agent Location field of the alert.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setAgentLocation`** takes the following arguments:

Name	Required	Type	Description
`agentLocation`	Yes	String	Agent Location value to set.

setAgentTime

A Workflow Engine function that sets the agent_time of the event to current time if the field does not exist in the event, or is more than the offset seconds in the past/future.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setAgentTime`** takes the following arguments:

Name	Required	Type	Description
`offset`	Yes	Number	The maximum number of seconds in the past or future to allow for the agent time. Set to 0 for current time.

setAnsibleJob

A Workflow Engine function that sets the instance and job template rule set to use for Ansible automation requests. Checks the template name against your Ansible Automation integration for a matching job template name. If found, uses the rule set to generate the request payload. Otherwise, uses the default job template rules.

This function relates directly to configurations from your Ansible Automation integration.

setAnsibleJob typically precedes a sendToAnsible action in your workflow, which uses the payload this function generates.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is only available for automation alert and automation Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **setAnsibleJob** takes the following arguments:

Name	Required	Type	Description
instance	Yes	String	Name of your Ansible Automation integration instance.
jobTemplateName	Yes	String	Name of the template. Must match the Workflow Job Template Name in your Ansible Automation integration.

Example

The following example demonstrates typical use of Workflow Engine function **setAnsibleJob**. It assumes you have set up the following:

- An Ansible Automation integration with the name "Ansible1".
- Within your Ansible Automation integration, a Workflow Job Template Name instance called "Restart-service".

Set the following:

- **instance**: Ansible1
- **jobTemplateName**: Restart-service
- Forwarding Behavior: Stop this workflow. This prevents further processing if the function fails to locate your configuration and returns **false**.

The UI translates your settings to the following JSON:

```
{"instance":"Ansible1","jobTemplateName":"Restart-service"}
```

setAutomationPayload

A Workflow Engine function that sets the automation solution, instance and Workflow Payload rule set to use for automation requests. Checks the Workflow Payload name against your automation integration for a matching job template name. If found, uses the rule set to generate the request payload. Otherwise, uses the default Workflow Payload rules.

This function currently supports the eyeShare and Ignio integrations and directly relates to configurations from these integrations.

setAutomationPayload typically precedes a sendToAutomation action in your workflow, which uses the payload this function generates.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is only available for automation alert and automation Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setAutomationPayload`** takes the following arguments:

Name	Required	Type	Description
`automationSolution`	Yes	String	Name of the automation solution. For example, `"Ignio"`.
`automationInstance`	Yes	String	Name of the integration instance. For example, `"Ignio1"`.
`payloadName`	Yes	String	Name of the payload. Must match the Workflow Payload Name in your integration.

Example

The following example demonstrates typical use of Workflow Engine function **`setAutomationPayload`**. It assumes you have set up the following:

- An eyeShare integration with the name `"eyeShare1"`.
- Within your eyeShare integration, a Workflow Payload instance where you have entered the Workflow Payload Name as `"Default"`.

Set the following:

- **`automationSolution`**: eyeShare
- **`automationInstance`**: eyeShare1
- **`payloadName`**: Default
- Forwarding Behavior: Stop this workflow. This prevents further processing if the function fails to locate your configuration and returns **`false`**.

The UI translates your settings to the following JSON:

```
{"automationSolution":"eyeShare","automationInstance":"eyeShare1","payloadName":
"Default"}
```

setClass

A Workflow Engine function that sets the class of the alert to a static value.

This function is available for alert and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setClass`** takes the following arguments:

Name	Required	Type	Description
class	Yes	string	Class to set for this alert.

setCoreEventField

A Workflow Engine function that sets a single core event field to a static value. For custom info, use the setCustomInfoValue or setCustomInfoJSONValue functions. For example set the **`agent_location`** field to `"London"`.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setCoreEventField`** takes the following arguments:

Name	Required	Type	Description
field	Yes	String	The field name other than **`custom_info`**.
value	Yes	Object	The static value to set.

Example

The following example demonstrates typical use of Workflow Engine function **`setCoreEventField`**.

```
{"field":"signature","value":"mySource:myClass:myType"}
```

setCustomInfoJSONValue

- A Workflow Engine function that adds or updates a custom info key to the specified JSON value.
- Accepts complex keys: a.b.c.d. The value must be a JSON object. Use the setCustomInfoValue function for to set string values.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setCustomInfoJSONValue`** takes the following arguments:

Name	Required	Type	Description
`key`	Yes	String	Custom info key for which to set the JSON. Complex keys are allowed. Do not include `"custom_info"` in the key.
`value`	Yes	Object	JSON value that you want to set.

Example

The following example demonstrates typical use of Workflow Engine function **`setCustomInfoJSONValue`**. If you want to set the JSON value for the **`custom_info.services`** key, set the following:

- key: services
- value: {"service_list":["Network","Database"]}

The UI translates your settings to the following JSON:

```
{"service_list":["Network","Database"]}
```

The Workflow Engine updates the object fields as follows:

```
 "custom_info":
    {"services":
        {"service_list": ["Network","Database"]}
    }
```

setCustomInfoValue

A Workflow Engine function that adds or updates a custom info key to a specified string value. Accepts complex keys: a.b.c.d. The value must be a text string. Use the setCustomInfoJSONValue for JSON object values.

This function is available for alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setCustomInfoValue`** takes the following arguments:

Name	Required	Type	Description
`key`	Yes	String	Custom info key for which to set the value. Complex keys are allowed.
`value`	Yes	String	Value to set. Must not be JSON.

setDescription

A Workflow Engine function that sets the description of the object. The action does not override manual descriptions.

This function is available for alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setDescription`** takes the following arguments:

Name	Required	Type	Description
`description`	Yes	String	Description to set.

setEnrichment

A Workflow Engine function that updates a single record in the enrichment datastore with data from an alert. Returns **`true`** if the request is successful.

This function relates directly to the API details from your Enrichment API.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for event, alert, and enrichment workflows.

This function does not modify the in-scope object when it updates enrichment data.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **setEnrichment** takes the following arguments:

Name	Required	Type	Description
attribute	Yes	String	Name of the attribute to lookup. For example, "hostname".
value	Yes	String	Name of the field or **workflowContext** key holding the data to lookup. To specify a **workflowContext** key, prefix with "workflowContext". For example, "workflowContext.lookupkey".
data	Yes	String	Name of the field or workflowContext key which holds the data to store against the source key. If you are using a workflowContext key, prefix with the string "workflowContext". For example, "workflowContext.datakey". Must contain a vaild JSON object.

Example

The following example demonstrates typical use of Workflow Engine function **setEnrichment**.

You want to send an update to your Enrichment API endpoint, using an attribute called "source" as the search key and the contents of the workflowContext key “data” as the enrichment data to store. Set the following:

Within your endpoint you have an attribute called "source". You want to send an update to the value of the **custom_info.lookupkey** field and use the contents of the workflowContext key “data” as the enrichment data to store. Set the following:

- **attribute**: source
- **value**: custom_info.lookupkey
- **data**: workflowContext.datakey

The UI translates your settings to the following JSON:

```
{"attribute":"source","value":"custom_info.lookupkey","data":"workflowContext.datakey"}
```

If successful, the function returns **true** and sends a request to the API endpoint, using the object source field as the search key.

setEnrichmentBulk

A Workflow Engine function that updates multiple records in the enrichment datastore with an array of data from an alert. Returns **true** if the request is successful.

This function relates directly to the API details from your Enrichment API.

This function is available as a feature of the Add-ons v1.4 download and later.

This function is available for event, alert, and enrichment workflows.

This function does not modify the in-scope object when it updates enrichment data.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **setEnrichmentBulk** takes the following arguments:

Name	Required	Type	Description
data	Yes	String	Name of the field or workflowContext key which holds the data to store against source key. To specify a **workflowContext** key, prefix with "workflowContext". For example, "workflowContext.datakey". Must contain a vaild array of JSON objects which contain the attribute, value, and enrichment values to use.

Example

The following example demonstrates typical use of Workflow Engine function **setEnrichmentBulk**.

You want to send an update to your Enrichment API endpoint using data stored in the workflowContext key **"data"** as the enrichment data to store. Set the following:

- **data**: workflowContext.datakey

The UI translates your settings to the following JSON:

```
{"data":"workflowContext.datakey"}
```

- The data must contain an array of JSON objects which contain the attribute, value and enrichment to store. For example:

```
[
  {
    "attribute": "source",
    "value": "node_1",
    "enrichment": { "service": "service_1"}
  },
  {
    "attribute": "source",
    "value": "node_2",
    "enrichment": { "service": "service_2"}
  }
]
```

This results in two update requests to the Enrichment API: one request to store the **{ "service": "service_1" }** enrichment data against the attribute **"source"** and value "node_1", and a second request to store the **{ "service": "service_2" }** enrichment data against the attribute **"source"** and value "node_2". If these requests are successful, the function returns **true** and applies the updates.

setExternalId

A Workflow Engine function that sets the external_id field of the event or alert.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **setExternalId** takes the following arguments:

Name	Required	Type	Description
externalId	Yes	String	external_id value to set.

setManager

A Workflow Engine function that sets the Manager field of the event or alert.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **setManager** takes the following arguments:

Name	Required	Type	Description
manager	Yes	String	Manager value to set.

setPuppetAutomation

A Workflow Engine function that sets the instance and job template rule set to use for Puppet automation requests. Checks the template name against your Puppet integration for a matching job template name. If found, uses the rule set to generate the request payload. Otherwise, uses the default job template rules.

This function relates directly to configurations from your Puppet.

setPuppetAutomation typically precedes a sendToPuppet action in your workflow, which uses the payload this function generates.

This function is available as a feature of the Add-ons v1.3 download and later.

This function is only available for automation alert and automation Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **setPuppetAutomation** takes the following arguments:

Name	Required	Type	Description
instance	Yes	String	Name of your Puppet integration instance.
templateName	Yes	String	Name of the template. Must match the Workflow Job Template Name in your Puppet integration.

Example

The following example demonstrates typical use of Workflow Engine function **setPuppetAutomation**. It assumes you have set up and configured the following:

- A Puppet integration with the name "Puppet1".
- Within your Puppet integration, a Workflow Job Template Name instance called "my-plan".

Set the following:

- **instance**: Puppet1
- **templateName**: my-plan
- Forwarding Behavior: Stop this workflow. This prevents further processing if the function fails to locate your configuration and returns **false**.

The UI translates your settings to the following JSON:

```
{"instance":"Puppet1","templateName":"my-plan"}
```

setSeverity

A Workflow Engine function that sets the severity of the alert.

This function is available for alert and enrichment workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **setSeverity** takes the following arguments:

Name	Required	Type	Description
severity	Yes	Number	Severity value to set for this alert. See Severity Reference for a list of severity values.

setSituationFlag

A Workflow Engine function that sets a flag for a Situation. If the Situation already has a flag set, using this action replaces it.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **setSituationFlag** takes the following arguments:

Name	Required	Type	Description
flag	Yes	String	Flag to set.

Example

The following example demonstrates typical use of Workflow Engine function **setSituationFlag**.

If you want to set the Situation's flag to "TICKETED", enter the following:

- **flag**: TICKETED

The UI translates your settings to the following JSON:

```
{"flag":"TICKETED"}
```

Given a Situation with the following flag:

```
{
    "situationFlags": [
        "TICKET_PENDING"
    ]
}
```

The Workflow Engine updates the object as follows:

```
{
    "situationFlags": [
        "TICKETED"
    ]
}
```

setSituationState

A Workflow Engine function that sets the state of the Situation. Not to be confused with Situation status.

This function is available for Situation workflows only.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setSituationState`** takes the following arguments:

Name	Required	Type	Description
state	Yes	String	State to set, for example TICKETED.

setSource

A Workflow Engine function that sets the source (hostname) field of the event or alert.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`source`** takes the following arguments:

Name	Required	Type	Description
`source`	Yes	String	Source to set for the event or alert.

setSourceId

A Workflow Engine function that sets the source_id field of the event or alert.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setSourceId`** takes the following arguments:

Name	Required	Type	Description
`sourceId`	Yes	String	source_id to set for the event or alert.

setType

A Workflow Engine function that sets the type of the alert.

This function is available for alert and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`setType`** takes the following arguments:

Name	Required	Type	Description
type	Yes	String	Type to set for this alert.

Example

The following example demonstrates typical use of Workflow Engine function **`setType`**. If you want to set the **`type`** for an object to `"availablity"`, enter the following:

- type: availability

The UI translates your settings to the following JSON:

```
{"type":"availability"}
```

sigActionFilter

A Workflow Engine function that returns **`true`** if the Situation action matches the specified type. Operates as a filter that stops processing Situations.

This function accepts an array Situation action types. See Situation Action Codes for a list. Specify which actions you want to continue processing, and use the either the `"Stop This Workflow"` or `"Stop All Workflows"` forwarding behavior to stop processing any other actions.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **`sigActionFilter`** takes the following argument:

Name	Required	Type	Description
`actionType`	Yes	Array	See Situation Action Codes for a list of Situation actions. For example `["Situation Updated"]`.

sigActionToolFilter

A Workflow Engine function that returns **`true`** if the specified tool has been run against a Situation. For example, a ticketing integration tool.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **sigActionToolFilter** takes the following argument:

Name	Required	Type	Description
toolName	Yes	String	Name of the tool

simpleLookup

A Workflow Engine function that defines the simple lookup as two arrays of equal length: keys and values. When the value of fromField matches a value in the keys array, sets toField to the value in the values array with the corresponding index.

This function is intended to make administration and usage easier, and is designed for short lists rather than for long lookups. For longer, more complex lookups, use the staticLookup function, which uses a configuration file.

This function is available as a feature of the Workflow Engine v1.2 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **simpleLookup** takes the following arguments:

Name	Required	Type	Description
fromField	Yes	String	Source field of the key.
keys	Yes	Object	Array of keys as a JSON array.
values	Yes	Object	Array of values as a JSON list.
toField	Yes	String	Destination field. Overwrites any existing values.

Example

The following example demonstrates typical use of Workflow Engine function **simpleLookup**, in which you perform a simple lookup that translates a textual severity in an event to a number-based severity.

Given this mapping of textual severity to it's numeric equivalent:

```
"clear" : 0
"unknown" : 1
"warning" : 2
"minor" : 3
"major" : 4
"critical" : 5
```

If you take your "key" from **custom_info.sourceSeverity** and put the looked up value into "severity", set the following:

- **fromField**: custom_info.sourceSeverity
- **keys**: ["clear","unknown","warning","minor","major","critical"]
- **values**: [0,1,2,3,4,5]
- **toField**: severity

The UI translates your settings to the following JSON:

```
{
"fromField":"custom_info.sourceSeverity",
"keys":["clear","unknown","warning","minor","major","critical"],
"values":[0,1,2,3,4,5],
"toField":"severity"
}
```

The action returns **true** if the **fromField** value is found in the "keys" and the corresponding "value" was successfully set in **toField**.

The action returns **false** if the **fromField** has no value or was not found in the "keys" , the value was not successfully set, or if the "keys" and "values" are not of equal length.

situationDelta

A Workflow Engine function that returns **true** when attributes have changed. This is based on the **previous_data** metadata, which Cisco Crosswork Situation Manager sends with the situation object in a situationUpdate event.

Only use this function in conjunction with an entry filter that includes the **event_handler** trigger for "Situation Updated".

This function does not check the values of the attributes, only if the attributes have changed. As standard de-duplication changes attributes, use this function carefully.

Cisco recommends placing **situationDelta** in an engine dedicated to handling Situation Updates and other alert event handlers. This prevents updated alerts re-entering the processing chain through standard Situation Workflows. Contact your Cisco Crosswork Situation Manager administrator for more information.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **alertDelta** takes the following arguments:

Name	Required	Type	Description
fields	Yes	Object	List of attributes to check for change. Accepts granular custom info attributes.

Example

The following example demonstrates typical use of Workflow Engine function **situationDelta**.

You want to check if the moderator of a Situation has changed before performing subsequent actions in your workflow. You could use an entry filter to check for a specific moderator, but in this instance the value of the moderator is not relevant, only that it has changed.

Using a separate Workflow Engine to prevent unwanted re-entry, you set up a workflow with an entry filter that includes the **event_handler** trigger for "Situation Update" and the moderator as "Unassigned":

(event_handler = "Situation Update") AND (moderator != "anon")

Set the following:

- **fields**: moderator
- Forwarding behavior: Stop this workflow. This ensures that if the alert owner has not changed, subsequent actions in this workflow do not execute.

The UI translates your settings to the following JSON:

```
{"fields":["moderator"]}
```

If the Situation metadata shows that the "moderator" has changed, the function returns **true** and the alert is forwarded to the next action in the workflow.

If function does not detect a change of ownership, the function returns **false** and the forwarding behaviour prevents subsequent actions in the workflow from executing.

skip

A Workflow Engine function that forwards an in-scope object to the next chained moolet using the standard forwarding mechanism, and skips the rest of the workflows in the current engine. This is useful if you have an engine with many workflows. For example, you may only want to process the workflow from the first matching entry filter for performance reasons.

You may also want to use this function to ensure no further actions execute after the first workflow. For example, if a lower action has a more open entry filter.

This function is available as a feature of the Workflow Engine v1.1 download and later.

This function is available for event, alert, enrichment, and Situation workflows.

This function is only compatible with the "Stop All Workflows" Forwarding Behavior, and the function always returns **false**.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **skip** has no arguments.

Example

To use the **skip** function, set it as the last action in your workflow and ensure Forwarding Behavior is set to "Stop All Workflows".

staticLookup

A Workflow Engine function that searches for a **key** in a static lookup table, retrieves the corresponding value, and applies that value to a **field** in the object. **lookupName** references a .lookup file in JSON format in the following folder: **$MOOGSOFT_HOME/config/lookups/**.

For example, **Locations** refers to **$MOOGSOFT_HOME/config/lookups/Locations.lookup**. On first use, the lookup loads into constants. You do not need to edit the Workflow Engine Moobot to load. The default lifespan for the lookup is 3600 seconds, after which the Workflow Engine reloads the file.

This function is available for event, alert, enrichment, and Situation workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **staticLookup** takes the following arguments:

Name	Required	Type	Description
key	Yes	String	Source field to use as the key.
lookupName	Yes	String	Name of the lookup. Corresponds to a lookup file in **$MOOGSOFT_HOME/config/lookups/lookupName.lookup**.
field	Yes	String	Field to set the result of the lookup to. If the lookup is unsuccessful, this is set to null or if there is a key named 'default' the values are taken from that.
lifespan	Yes	Number	Lifespan of the current lookup data in memory before the Workflow Engine reloads the data from disk. Default is 3600 seconds.

stripFQDN

A Workflow Engine function that splits a fully qualified domain name (FQDN) into a hostname/short name and a domain name and updates fields with the values.

If **shortNameField** begins with "www" or a derivative, sets the value to the subsequent segment of the domain. For instance, "www3.example.com" returns "example'.

If you don't want to map the domain name, enter null or an empty string, "", for the **domainNameField** .

This function is available for event, alert, and enrichment workflows.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **stripFQDN** takes the following arguments:

Name	Required	Type	Description
fqdnField	Yes	String	Name of the field to parse the FQDN.
shortNameField	Yes	String	Destination field for the extracted short name/host name.
domainNameField	No	String	Destination field for the extracted domain name.

upperCase

A Workflow Engine function that changes the value of a field to uppercase. For example, changes a value of "Network" to "NETWORK".

This function is available for event, alert, enrichment, and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **upperCase** takes the following argument:

Name	Required	Type	Description
field	Yes	String	The name of the field.

willCreateNewAlert

A Workflow Engine function that returns **true** if the event will create a new alert.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **willCreateNewAlert** has no arguments.

willDeduplicateAlert

A Workflow Engine function that returns **true** if the event will deduplicate into an existing alert.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference.Workflow Engine Functions Reference

Arguments

Workflow Engine function **willDeduplicateAlert** has no arguments.

Troubleshoot the Workflow Engine

The Workflow Engine is a flexible tool that can help you transform your data and manage its flow through Cisco Crosswork Situation Manager data processing. Workflow Engine functions provide you programmatic control over your data, but sometimes your workflows may not behave the way you expect.

This topic contains ideas to help you debug your workflows and get the most from the Workflow Engine.

Troubleshoot from the UI

Consider the following options for troubleshooting workflows from the Cisco Crosswork Situation Manager UI:

- If you have multiple workflows enabled, but one of them is behaving unexpectedly, try temporarily disabling the other workflows to see if it works on its own. If so, reactivate the other workflows one by one, testing at each step to see if one of the other workflows is affecting it.
- Check the forwarding behavior for your workflow actions. The forwarding behavior controls subsequent processing when the function returns **false**. Stop This Workflow prevents the object passing to subsequent actions and Stop All Workflows prevents the object from passing to any subsequent action or workflow.

- Test your entry filter for the workflow. If your objects are not meeting entry filter requirements, the workflow will not process them.
- Verify your source fields and destination fields. Make sure that the names match up exactly. If you are using complex keys, make sure that you have the path exactly right. For example: **`custom_info.eventDetails.services`**.
- If you are using an event that specifies a Moolet, check the Moolet name under Settings > Self Monitoring > Event Processing. For example, "Default Cookbook".

Troubleshoot from the moog_farmd.log

If you have access to the log for Moogfarmd, you have a lot more troubleshooting options to identify exactly what is happening with your objects as they progress through workflows.

To enable debug logging for Moogfarmd, execute the following:

```
farmd_cntl --loglevel debug
```

When you are through logging, reset the log level to warn:

```
farmd_cntl --loglevel warn
```

You can find the Moogfarmd log at **`/var/logs/moogsoft`**. See Configure Logging for more information.Configure Logging

The Workflow Engine includes the following logging functions to help you troubleshoot:

- logMessage: Logs a message to the Moogfarmd log.
- logWorkflowDuration: Logs debug messages for the workflow execution duration.

The log messages from the Worfklow Engine include the engine name along with details about the object processing in the workflow. This means that you can use the **`tail`** command to observe the activity within an engine. For example:

```
tail -f MOO.moog_farmd.log | grep ":Alert.Workflows"
```

Within the log output, you can search for specific things, including:

- The function name you are troubleshooting.
- Identifying data for the object you are processing, such as the event signature.
- Identifying information about an entry or sweep up filter.

See Example Workflow Engine log for sample messages and their meanings within the log context.

Example Workflow Engine log

The following log segment includes comments to highlight the different aspects of a Workflow Engine log:

```
### Alert did not pass the entry filter ###
DEBUG: [3:Enrichment Workflows][20191002 16:24:55.983 -0400]
[CWorkflow.java:470] +|Moolet [Enrichment Workflows] - workflow [Closed Alerts
Filter]: message [{"Elements":{active_sig_list=[67, 68], agent=DATA_SOURCE,
agent_location=my_agent_location, alert_id=165, class=my_class, count=3,
custom_info={eventDetails={agent=TestAgent1, first_occurred=1570047828,
service=SAP, name=REST LAM Post 1, team=SAP Support}}, description=DESC: Host 1
Sig 1, entropy=0.8312803355385304, event_id=2899, external_id=my_external_id,
first_event_time=1570047828, int_last_event_time=1570047828,
```

```
last_event_time=1570047896, last_state_change=1570047828, manager=TestMgr1,
owner=2, rc_probability=null, severity=5, sig_list=[67, 68],
signature=lnux100:sig1, significance=3, source=lnux100,
source_id=192.168.100.101, state=2, type=TestType1}, "Topic":"alerts",
"Seq":"0", "SessId":"4769192054476008521", "Pdu":"E_MooMsg",
"MessageId":"c2fc745a-8572-4982-a012-69fe64b84e96", "CorrelationId":"ff62fbbb-
45ff-44f8-a1b5-9341bf33e729", "Metadata":{action=Event Added To Alert,
clock_time=1570047895, message_type=1,
previous_data={last_event_time=1570047869, count=2}, user_id=2},
"UsedCount":"null", "AckPoint":"0"}] failed to pass entry filter [state = 9].|+

### Workflow is inactive ###
DEBUG: [3:Enrichment Workflows][20191002 16:24:55.983 -0400]
[CWorkflow.java:463] +|Moolet [Enrichment Workflows] - workflow [Enrich From
SNow] inactive, sending message to the next Workflow/Moolet.|+

### Active workflow begins ###
DEBUG: [3:Enrichment Workflows][20191002 16:24:55.983 -0400]
[CWorkflow.java:294] +|Moolet [Enrichment Workflows] - workflow [Test External
DB]: starting delay of [0] seconds for msg [{"Elements":{active_sig_list=[67,
68], agent=DATA_SOURCE, agent_location=my_agent_location, alert_id=165,
class=my_class, count=3, custom_info={eventDetails={agent=TestAgent1,
first_occurred=1570047828, service=SAP, name=REST LAM Post 1, team=SAP
Support}}, description=DESC: Host 1 Sig 1, entropy=0.8312803355385304,
event_id=2899, external_id=my_external_id, first_event_time=1570047828,
int_last_event_time=1570047828, last_event_time=1570047896,
last_state_change=1570047828, manager=TestMgr1, owner=2, rc_probability=null,
severity=5, sig_list=[67, 68], signature=lnux100:sig1, significance=3,
source=lnux100, source_id=192.168.100.101, state=2, type=TestType1},
"Topic":"alerts", "Seq":"0", "SessId":"4769192054476008521", "Pdu":"E_MooMsg",
"MessageId":"c2fc745a-8572-4982-a012-69fe64b84e96", "CorrelationId":"ff62fbbb-
45ff-44f8-a1b5-9341bf33e729", "Metadata":{action=Event Added To Alert,
clock_time=1570047895, message_type=1,
previous_data={last_event_time=1570047869, count=2}, user_id=2},
"UsedCount":"null", "AckPoint":"0"}]|+

### Name of the function that is processing ###
DEBUG: [3:Enrichment Workflows][20191002 16:24:55.984 -0400]
[CWorkflowBotAction.java:196] +|Performing action [enrichOneToOne]|+

### Depending on the function, different logs here ###

###  Alert updated ###
DEBUG: [3:Enrichment Workflows][20191002 16:24:56.096 -0400] [CMooMsg.java:1086]
+|Encoded size [991] json[{"_MOOTADATA_":{"action":"Alert
Updated","clock_time":1570047896,"message_type":1,"previous_data":{"custom_info"
:{"enrichment":null,"eventDetails":{}},"last_state_change":1570047828}},"active_
sig_list":[67,68],"agent":"DATA_SOURCE","agent_location":"my_agent_location","al
ert_id":165,"class":"my_class","count":3,"custom_info":{"eventDetails":{"agent":
"TestAgent1","first_occurred":1570047828,"service":"SAP","name":"REST LAM Post
1","team":"SAP
Support"},"enrichment":{"ci":{"Name":"lnux100","AssetClass":"Linux
Server"}}},"description":"DESC: Host 1 Sig
1","entropy":0.8312803355385304,"external_id":"my_external_id","first_event_time
":1570047828,"int_last_event_time":1570047828,"last_event_time":1570047896,"last
_state_change":1570047896,"manager":"TestMgr1","owner":2,"rc_probability":null,"
severity":5,"sig_list":[67,68],"signature":"lnux100:sig1","significance":3,"sour
```

```
ce":"lnux100","source_id":"192.168.100.101","state":2,"type":"TestType1"}]|+

### Action completing with an exit status of 'true' ###
DEBUG: [3:Enrichment Workflows][20191002 16:24:56.103 -0400] [CMDB-WFE.js:403]
+|Enrichment Workflows::enrichOneToOne: Exiting action with a status of true|+

### Workflow Finished and sending to next Moolet ###
DEBUG: [3:Enrichment Workflows][20191002 16:24:56.104 -0400]
[CPassToNextMoolet.java:63] +|Moolet [Enrichment Workflows] - Sending message to
the next Moolet|+
DEBUG: [3:Enrichment Workflows][20191002 16:24:56.104 -0400]
[CMsgDispatch.java:516] +|Dispatching message from [Enrichment Workflows]|+

### Name of the next Moolet for the alert ###
DEBUG: [3:Enrichment Workflows][20191002 16:24:56.104 -0400]
[CMsgDispatch.java:547] +|Dispatching to [MaintenanceWindowManager]|+
```

Workflow Engine Strategies and Tips

The Cisco Crosswork Situation Manager Workflow Engine is a powerful tool that gives you access to your event, alert and Situation data and lets you control data processing flow. This topic covers some strategies to help you get the most out of the Workflow Engine.

Before you begin

Taking some time to prepare before you start creating workflows in Cisco Crosswork Situation Manager can help improve your experience. The following include some suggestions to help you get ready:

- Take time to define outcomes you want to accomplish with the Workflow Engine. Read through the example use cases listed in the Workflow Engine topic to gather ideas.
- Pick one use case and think about the supporting data that can help you define your workflow. For example, you want to stop processing of clear, severity 0, events if they are going to create a new alert. In this case, you need the **`severity`** field. For more information see Alert and Event Field Reference.
- Look through the Workflow Engine Functions Reference to find the right function you need to accomplish your task. For example, to see if an event will create a new alert, you can use willCreateNewAlert.
- If possible, set up a REST LAM so you can send sample data to test out your workflows. See REST LAM. You can use cURL or a graphical API client to send event data to the REST LAM.
- If possible, get SSH access to the machine running Moogfarmd so you can access the log for troubleshooting. See Troubleshoot the Workflow Engine for more information.

Creating workflows

Workflows are containers for a set of actions to process your data. In a complex system, you may create hundreds of workflows to handle all kinds of scenarios. Consider the following as you define your workflows:

- Workflow Engines process data through individual workflows in the numeric order from the UI. If possible, work with one active workflow at a time while you design and configure your workflow. This way you can focus on a single behavior without worrying about an upstream or downstream workflow.
- Whenever possible, use an entry filter to limit the number of objects entering your workflow. Running a workflow takes processing power and time. Using an entry filter ensures that only pertinent objects pass through a workflow. Entry filters are also more performant than action based

filters. For example, if I want to stop processing clear events, I can create a filter: **`severity = "Clear"`**.

- If you want to use a sweep up filter to process multiple alerts or Situations, verify that the function you plan to use works with sweep up filters. You can check the Workflow Engine Functions Reference or the topics for individual functions.
- When objects enter a workflow as part of a sweep up filter, the workflow processes each action on all objects in turn. For example, it executes action one for all objects before proceeding to action two.
- When you enable multiple workflows, consider the impact of upstream workflows on downstream ones. For example, keep the default Closed Alerts Filter early in the workflow chain. This prevents those events from entering later workflows. Also consider the impact of event workflows on alert and Situation workflows.

See Manage Workflows for steps to create a workflow.

Adding Actions to workflows

You add actions to a workflow to manipulate data and control the data processing flow for events, alerts, enrichment, and Situations. Consider the following as you add actions to your workflow:

- The UI validates that the data you enter for function arguments as JSON. Where possible, it checks validity of number and string data. For example 0-5 are the only valid severities. It does not check for valid field names or value data.
- Unless instructed, you do not need to enter quotation marks around values, except strings where they represent elements in an array.
- The Workflow Engine uses JavaScript regular expression notation. Consider using a third-party regular expression validator to help with syntax. For example, regex101. Cisco does not endorse any specific tool. Before using any tool, verify that it meets your organization's standard for security and privacy.
- The forwarding behavior applies when the action returns **`false`**. For more detail, see Action types and forwarding behavior.
- Actions execute in numeric order. Test your workflow for both negative and positive cases with the addition of each new action.

See Manage Workflow Engine Actions for steps to add actions to a workflow.

Action types and forwarding behavior

The Workflow Engine has several types of actions to choose from. Each action lets you set the forwarding behavior to control the downstream flow of the object in the case that the action returns **`false`**. The type of action should influence how you choose your forwarding behavior. Action types include:

- Conditionals that return **`true`** or **`false`** to let you control whether or not to proceed with the current object. You can use these as an "action" based filter if you need to filter data within a workflow after the execution of previous actions. For maximum efficiency, you may consider breaking a workflow into multiple workflows and using the entry filter for the subsequent workflow.

- Transformers that update object data. In general transformer actions should Always Forward unless doing so compromises the workflow. For example, when a failed action impacts the subsequent action.
- Data actions that action retrieved data and pass it to a consecutive action or that send data asynchronously to an external data sink. Treat data retrievers like conditionals. For example, stop the workflow if it returns **`false`** since subsequent actions may not run. An asynchronous exporter always returns **`true`** regardless of the result from the asynchronous return.