Cisco Crosswork Situation Manager Add-ons

Add-on documentation for Cisco Crosswork Situation Manager.

Abstract

This draft does not include associated linked content. Refer to the Cisco Crosswork Situation Manager documentation for details.

Table of Contents

Cisco Add-ons

Cisco periodically releases add-ons to extend and enhance the core Cisco Crosswork Situation Manager functionality. For example, new Workflow Engine functions, new Worfklow Engines, or Integrations tiles.

If you haven't already, you need to create the Enrichment API data store to use the Enrichment API Integration. See Create the Enrichment API Data Store for instructions. If you upgraded from Cisco Crosswork Situation Manager v.7.3.0 and were using the Integrations API with Cisco Add-ons v.1.4.0 or later, you do not need to create the Enrichment API data store.

Cisco Crosswork Situation Manager v8.0 is compatible with v2.0.0 and later only.

All add-ons releases are cumulative and include the fixes from previous releases.

You can download the current add-ons (`Crosswork-Situation-Manager-Add-ons-2.2.0.tar.gz`) from Cisco eDelivery.

See Install Cisco Add-ons [12] for installation details.

Add-ons v2.2

If you haven't previously done so, Create the Enrichment API Data Store to use the Enrichment API Integration.

Cisco Add-ons v2.2.0 includes the following updates:

- ADD-36: Adds the AlertOps Integration.
- ADD-43: Improvements for the PagerDuty Integration.
- ADD-44: Adds the Microfocus Operations Orchestration Automation Integration.
- ADD-106: Adds WFE Integration framework.
- ADD-101: Bug fixes for the Email Integration.
- WFE-322: Fixes an issue so that WFE function addTopologyNode correctly updates descriptions.
- WFE-325: Fixes an issue to so that actions perform a moogdb.reload() when appropriate.
- WFE-330: Fixes $TOLIST() so that it cannot map to an empty item.
- WFE-138: Adds functions hasSweptUp and hasNotSweptUp.
- WFE-172: Adds support for Workflow Job Templates to Ansible Automation Integration.
- WFE-194: Adds WFE support for the Microfocus Operations Orchestration Integration Automation.
- WFE-203: Adds removeItemsFromList function.
- WFE-258: Adds compareFields function.
- WFE-296: Adds addRestHeader function.
- WFE-297: Adds functions to support integration triggers.
- WFE-301: Adds addLocationData function.
- WFE-317: Updates WFE functions to use updateClosedAlert and updateClosedSituation as appropriate.
- WFE-320: Adds addToContext function.
- WFE-321: Adds listSituationHosts function.
- WFE-324: Updates the WFE function getPayload to encapsulate the payload within an array.
- WFE-326: Adds listSituationAlertIds function.
- WFE-327: Adds setSituationServices function.

Add-ons v2.1

If you haven't previously done so, Create the Enrichment API Data Store to use the Enrichment API Integration.

Cisco Add-ons v2.1.0 includes the following updates:

- ADD-1: Function and integration tile to send emails.
- ADD-16: Situation moderator updates.
- ADD-20: Slack Integration can send notifications to Slack channels.
- WFE-223: `getCorrelationInfo` - checks that correlation info exists for a Situation.
- WFE-233: The `moog_enrichment_util.sh` tolerates "invisible" windows carriage returns in input files.
- WFE-240: Substitution macros: `$PRC`, `$REF`, `$TYPE`.
- WFE-247: `sendAlertsToWorkflow` - Sends situation alerts to a workflow.
- WFE-251: Improves Topology API WFE function logic.
- WFE-253: Improves reliability with substitution and validation of parameters.
- WFE-256: `isPrimaryTeamSet` - returns true if a Situation's primary team is set.
- WFE-257: `basicMaths` - writes the result of basic maths (+, -, * , / , %) on two fields to either `custom_info` or the workflow context.
- WFE-263: Substitution macros: `$EXPAND_AS_CSV`, `$EXPAND_AS_STRING`, `$TO_LIST`.
- WFE-264: `getThreadEntry` - retrieves the corresponding thread entry if a Situation contains a thread `entry_id` and stores it under the `threadEntry` workflowContext key.
- WFE-265: Conversion Maps with function `convertField`.
- WFE-266: `addCorrelationInfo` - adds an External ID to a Situation.
- WFE-267: Enrichment API `moog_enrichment_util` scripts have proxy settings.
- WFE-268: Fixes the `checkTopologyyLink` function.
- WFE-276: `convertStringToList` - converts a string containing regular separators (e.g. a comma) into a list (a JavaScript Array) and copies the resulting list into a named target or overwrites the value of the original string with the new list.
- WFE-277: Substitution and validation of params now correct in workflow function template.
- WFE-284: `IsNotClear` function now works as expected.
- WFE-285: `dejaVu` – checks for repeated data.
- WFE-286: Catasuraus now uses correct log message levels.
- WFE-304: JDBC and ServiceNow Enrichment Integrations no longer call `constants.reload()`.

Add-ons v2.0.0

Cisco Add-ons v2.0.0 introduces several new functions and fixes to existing functions and Integrations tiles:

- WFE-126: Preset PRC values for topology based situations (or any situation).
- WFE-162: Adds Create topology function.
- WFE-163: Adds a function to create a node.
- WFE-164: Adds a function to create a link between two nodes.
- WFE-165: Adds a function to clone a topology.
- WFE-166: Adds a function to delete an active or inactive topology.
- WFE-167: Adds a function to remove a node from a topology.
- WFE-168: Adds a function to delete a topology link.
- WFE-169: Adds a function to check a topology exits.
- WFE-170: Adds a function to check a node exists.
- WFE-171: Adds a function to check a link exists.
- WFE-177]: Adds populateNamedTopology function.
- WFE-195: Adds a function to activate a topology.
- WFE-196: Adds a function to deactivate a named topology.
- WFE-198: Adds an action to delete a node.
- WFE-199: Adds an action to delete a topology.
- WFE-213: Updates the Ansible Automation Integration for 8.0.
- WFE-214: Updates the Puppet Automation to 'Automation' for 8.0
- WFE-215: Updates the AyeHu eyesShare Automation for 8.0.
- WFE-216: Updates the Ignio Automation for 8.0.
- WFE-217: Updates ServiceNow Enrichment for 8.0.
- WFE-218: Updates JDBC Enrichment for 8.0.
- WFE-222: Corrects the description of sigActionFilter.
- WFE-224: Updates the Enrichment API for 8.0.
- WFE-225: Updates configuration for Integration Tiles for Enterprise 8.0 Release.
- WFE-226: Adds Eventless processing for Automation actions.
- WFE-227: Escapes of data in MySQL queries used by Enrichment API.
- WFE-236: Adds a dnsLookup function.
- WE 237: Adds a logCEvent function.

Install Cisco Add-ons

Cisco periodically provides updates to Cisco Crosswork Situation Manager as add-ons. Add-ons may comprise updates to the Workflow Engine, new Workflow Engine functions, integrations tiles, and other features.

This topic tells you how to install the latest version of the Cisco Add-ons. For information on the latest add-ons, see Cisco Add-ons [8].

If you haven't already, you need to create the Enrichment API data store to use the Enrichment API Integration. See Create the Enrichment API Data Store for instructions. If you upgraded from Cisco Crosswork Situation Manager v.7.3.0 and were using the Integrations API with Cisco Add-ons v.1.4.0 or later, you do not need to create the Enrichment API data store.

Before you begin

Before you install Cisco Add-ons:

- Verify you have SSH access to to your Cisco Crosswork Situation Manager machines. For distributed or Highly Available installations, update the add-ons on all core role machines where you run Moogfarmd and on the machines where you run the UI. See Server Roles.
- Download the latest add-ons bundle and transfer it to the machines where you are performing the update.
- Verify the credentials for the operating system user that runs Moogfarmd or the UI and perform all steps as that same user.
- The Add-ons include the following new Workflow Engines:
  - Situation Delta: `situation_delta_workflows.conf`
  - Alert Actions: `alert_actions_workflows.conf`
  - Alert Integration: `alert_integration_workflows.conf`
  - Situation Integration: `situation_integration_workflows.conf`.

  If you have any existing custom engines with the same name, you must rename the custom engines as part of the upgrade.

Download Cisco Add-ons

You can download the current add-ons (`Crosswork-Situation-Manager-Add-dons-2.2.0.tar.gz`) from Cisco eDelivery.

Install add-ons

To install add-ons, you add them to machines running Moogfarmd and the UI components. This procedure replaces existing components.

1. Create a backup of the moobots, integrations, and other add-on files on the instance. For example:

   ```
   tar -czf $MOOGSOFT_HOME/addons_backup.tar.gz -C $MOOGSOFT_HOME \
   bots/moobots \
   config \
   contrib \
   etc/integrations
   ```
2. Make a copy of `SimilarSigConfig.conf`:

   ```
   cp $MOOGSOFT_HOME/config/SimilarSigConfig.conf $MOOGSOFT_HOME
   ```
3. Extract the add-ons to `$MOOGSOFT_HOME`:

   ```
   tar -xzhf Crosswork-Situation-Manager-Addons-2.2.0.tar.gz -C
   $MOOGSOFT_HOME
   ```
4. Move `SimilarSigConfig.conf` back into place:

   ```
   mv $MOOGSOFT_HOME/SimilarSigConfig.conf $MOOGSOFT_HOME/config
   ```
5. Edit `$MOOGSOFT_HOME/config/moog_farmd.conf`. In the Moolets section, add include statements for the new Moolets to the list of existing Moolets. If you changed the name of any custom Moolet, update the name in the imports.

   ```
   ,
           {
             include : "alert_actions_workflows.conf"
           },
           {
             include : "situation_delta_workflows.conf"
           },
           {
             include : "situation_integration_workflows.conf"
           },
           {
             include : "alert_integration_workflows.conf"
           }
   ```

   Make sure to include the leading comma "," when you append the new Moolets to the list.
6. On core role machines, restart Moogfarmd:

   ```
   service moogfarmd restart
   ```

   You do not need to restart Nginx or Apache Tomcat on UI machines.

Conversion Maps

You can use this integration to create maps comprised of key-value pairs in the form of `name:alias`. The maps work with the `convertField` Workflow Engine function to look up the "name" and update the value of a field with the "alias".

The conversion map configuration follows a similar pattern as the configuration for LAM Mapping,

Before You Begin

Before you start to set up your integration, ensure you know the following about your conversion map:

1. The map name.
2. The keys and values you want create mappings for.

Configure the Conversion Map Integration

You can use this integration to configure multiple conversion maps. To complete the configuration, you must provide a unique integration name and have at least one conversion map that contains at least one mapping.

Integration Name

Name of the conversion map integration instance.

Type	String
Required	Yes
Default	MyMap

Conversion Name

Specifies the name of your conversion map.

Type	String
Required	No
Default	conversionName

Enable this Conversion

Enable the conversion for use in mapping.

Type	Boolean
Required	Yes
Default	Unchecked

Match Names with Case Sensitivity

Set the match type to case sensitive. Defaults to case insensitive.

Type	Boolean
Required	Yes
Default	Unchecked

Either Retain the Original Value, Set a Default, or Exclude

Specifies the behavior to perform if there is no match.

Type	One of **retain**: leave the original value unchanged **default**: return the default value **exclude**: actions using this map fail
Required	Yes
Default	retain

Name

Specifies the keys you wish to lookup.

Type	String
Required	Yes
Default	<item name>

Alias

Specifies the values you wish to return.

Type	String
Required	Yes
Default	<item alias>

Use a Conversion Map in a Workflow

You can use the conversion maps you configure in your workflows. For example, you can use the `convertField` function to lookup and update a field in an alert or Situation. Remember to use the map name you specified the configuration for the map name in the Workflow Engine Action.

Learn more

Watch the following video to learn more about Conversion Maps:

https://player.vimeo.com/video/445706365

Email Endpoints

You can install the Email Endpoints integration to enable email notifications to user or group email addresses. The integration allows you to send emails for alerts or Situations.

You can use the integration UI to define email servers, message subject, and message body templates in the integration UI. The email server and message template configurations work with the following Workflow Engine functions:

- **sendEmail**: Uses the email server configuration from the Email Endpoints integration to send an email when the workflow trigger condition is met. You specify the email subject and message in the workflow `action.sendEmail`
- **sendEmailUsingTemplate**: Uses the email server configuration and template from the Email Endpoints integration to send an email when the workflow trigger condition is met.

Before You Begin

The Email Endpoints integration uses the Cisco Crosswork Situation Manager mailer module and supports TLS and non-TLS connections. Before you start to set up your integration, ensure determine the settings for your target email server. Settings can include:

1. Server name or IP address
2. TLS requirements
3. Port number
4. Connection credentials

Email Servers

Specify the Email servers for Cisco Crosswork Situation Manager workflows and add to the Email Endpoints instance configuration. See the Email Endpoints Reference [26]for a description of all properties. You must define at least one instance.

Configure Email Endpoints

You can use the connection settings you gathered in **Before You Begin** to configure the Email servers required for Cisco Crosswork Situation Manager to send emails using the Workflow Engine. See the email documentation for more information on configuration.

Email Endpoints Reference

This is a reference for the Email Endpoints integration [22].

Server Config Name

Specifies the reference name of the email server.

Type	String
Required	Yes
Default	Email server reference name

Hostname

Specifies the hostname or IP address of the email server.

Type	String
Required	Yes
Default	Hostname or IP address of the mail server

Port

Specifies the port used for the email server.

Type	Integer
Required	Yes
Default	0

Account

Specifies the email address of the account.

Type	String
Required	Yes
Default	N/A

Password

The password for the account.

Type	String
Required	No
Default	N/A

Encrypted

Encrypt email messages.

Type	Boolean
Required	No
Default	Unchecked

START_TLS

Use Opportunistic TLS.

Type	Boolean
Required	No
Default	Unchecked

USE_TLS

Connect to the the email server using TLS.

Type	Boolean
Required	No
Default	Unchecked

Use Proxy in Request

Specifies a connection for a proxy server if you want to connect to the external system through a proxy.

Type	Boolean
Required	Yes
Default	No

Proxy Host

Specifies the hostname for the proxy server.

Type	String
Required	If **Use Proxy in Request** is set to **Yes**.
Default	N/A

Proxy Port

Specifies the port number for the proxy server.

Type	Integer
Required	If **Use Proxy in Request** is set to **Yes**.
Default	8080

Proxy User

Specifies the user for the proxy server.

Type	String
Required	If **Use Proxy in Request** is set to **Yes**.
Default	dummy

Proxy Password

Specifies the password for the proxy server.

Type	String
Required	No
Default	N/A

Message Template Name

Specifies the name of the email template. You can use the name to differentiate different message templates.

Type	String
Required	Yes
Default	<message template name>

Message Subject

Specifies the subject field for a given email template.

Type	String
Required	Yes
Default	Enter message subject, macro substitution supported

Message Body

Specifies the body for a given email template.

Type	String
Required	Yes
Default	Enter message body, macro substitution supported - use $NL for newline

Team Name

The team name should match the team name resulting from `$EXPAND(teams)`. For an example, see sendEmailUsingTemplate [212].

Type	String
Required	Yes
Default	<team name>

Email Address

Email address destination for the team.

Type	String
Required	Yes
Default	<email address>

Slack v2.0

You can install the Slack Workspace integration to configure workflows to send notifications to Slack channels for alerts or Situations. The integration enables Slack notifications from Cisco Crosswork Situation Manager using the incoming webhook interface in Slack.

You can also use the Slack integration UI to define Slack workspaces and channel mappings based on the incoming webhooks configured in the Slack App.

This integration is available as a part of the Add-ons v2.1 and later downloads.

The Slack integration works with the following Workflow Engine functions:

- setSlackChannel: Sets the Slack instance, and channel for the workflow. The channel name is checked against the Cisco Crosswork Situation Manager Slack Workspace config. When it finds a matching channel name, it uses the associated incoming webhook in the generated request payload.
- notifySlack [183] : Sends a notification to the Slack channel in `setSlackChannel`.

Before You Begin

The Slack integration has been validated with the Slack Apps v2 API incoming webhook endpoint. Before you start to set up your Slack v2.0 integration, ensure you have met the following requirements:

- You have a Slack account and administrator privileges for your target Slack workspace.
- You have created a Slack channel for incoming messages from Cisco Crosswork Situation Manager.

Configure Slack

These instructions offer the basic information you need to configure Slack for integration with Cisco Crosswork Situation Manager. For more information on Slack, see the Slack documentation.

1. Sign-in to your target Slack workspace.
2. From the workspace dropdown menu in the left navigation pane, select **Settings & Administration > Manage apps**. If you are accessing your Slack workspace with the Slack App, a browser session opens.
3. In the browser session, click the **Build** button.
4. Click on **Create New App** or select the existing Slack App to add webhook endpoints to.
5. Select **Incoming Webhooks** from the left navigation pane.
6. Set **Activate Incoming Webhooks** to **On** if it isn't already.
7. Click **Add New Webhook To Workspace**. In the resulting dialog, select an existing workspace channel for the webhook.
8. Copy the new webhook URL from the **Incoming Webhooks** section on the App page. You'll use this URL to configure the integration.

Configure Slack Channels

You can configure the Slack channels to receive incoming webhooks from the Cisco Crosswork Situation Manager workflows and add them to the Slack Workspace integration configuration. You must define at least one instance.

1. Navigate to the **Integrations** tab.
2. Click Slack in the **Featured Integrations** section.
3. Provide a unique integration name. You can use the default name or customize the name according to your needs.
4. Configure the Slack integration for alerts or Situations to initiate and identify the payload you need. See the Slack v2 Integration Reference [33] for a description of all properties.
5. Define at least one instance to complete the configuration. If you are running multiple instances, identify any differences between them so you can configure these in the integration.

After you complete the configuration, Cisco Crosswork Situation Manager can send updates to Slack.

Slack v2 Integration Reference

This is a reference for the Slack v2 [29] integration.

Integration Name

Name of the Slack integration instance.

Type	String
Required	Yes
Default	Slack1

Wait for a Notification Request Response

Whether to wait for a response for requests.

Type	Boolean
Required	Yes
Default	No

Request Timeout in Seconds

Length of time (in seconds) to wait for a response from a request before returning a timeout.

Type	Integer
Required	Yes
Default	20

Use Proxy in Request

Specifies a connection for a proxy server if you want to connect to the external system through a proxy.

Type	Boolean
Required	Yes
Default	No

Proxy Host

Specifies the hostname for the proxy server.

Type	String
Required	If **Use Proxy in Request** is set to **Yes**.
Default	N/A

Proxy Port

Specifies the port number for the proxy server.

Type	Integer
Required	If **Use Proxy in Request** is set to **Yes**.
Default	8080

Proxy User

Specifies the user for the proxy server.

Type	String
Required	If **Use Proxy in Request** is set to **Yes**.
Default	dummy

Proxy Password

Specifies the password for the proxy server.

Type	String
Required	No
Default	N/A

Slack Workspace Name

The name of your target Slack workspace.

Type	String
Required	No
Default	<target workspace name>

Slack Channel Name

The name of your target Slack channel.

Type	String
Required	Yes
Default	<target channel name>

Slack Channel Incoming Webhook URL

URL of your Slack webhook.

Type	String
Required	Yes
Default	<general webhook>

Team Name

The team name should match the team name resulting from .`$EXPAND(teams)`.

Type	String
Required	No
Default	<team name>

Title Element

Type	String
Required	Yes
Default	text. Do not change.

Title Mapping Rule

The rule to generate the Slack message text.

Type	String
Required	Yes
Default	There has been a $EXPAND(internal_priority) level Situation

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 [58] 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:

- addDefaultCustomInfoValues [52]: Supersedes the existing `addDefaultValues` which used the now deprecated **Payload Maps** integration for the values.
- addDefaultValues [55]: Adds a set of default values to `custom_info` based on a payload map. Sweep up filter applies.
- addItemToList [58]: Adds an item or items to an array. Sweep up filter applies.
- addLocationData [56]: Populates the standard address fields in the standard `custom_info.location` object.
- addTags [60]: Adds or updates a custom info field called "tags" with an array of string values.
- addToContext [62]: Updates the workflow context with a key: value pair.
- appendFields [69]: Appends a concatenated set of fields to an existing field, using a separator character.
- appendString [70]: Appends a static string to an existing field separated by a space character.
- basicMaths [74]: Allows basic maths (+, -, * , / , %) to be performed on two fields that write the result to a destination field in either `custom_info` or the workflow context.
- ceventFilter [77]: Returns `true` if the object matches a SQL-like filter. Sweep up filter applies.
- checkSeverity [78]: Checks the severity level of the object.
- classifyEvent [83]: Sets the class, type, and severity fields of an event based upon its contents using a predefined classification algorithm.
- compareFields [88]: Returns true or false based on the comparison of two string or number values.
- concatFields [89]: Sets the value of a field to a string representing a set of concatenated fields.
- contextFilter [91]: Filters a `workflowContext` object for a specified name field. Sweep up filter applies.
- convertField [92]: Converts a field using mappings defined in the Conversion Maps integration tile.
- convertStringToList [96]: Converts a string of words separated by a separator character into a Javascript array (list).
- convertToJSON [97]: Converts the object to JSON and adds it to the workflowContext for use in subsequent actions.
- copyFieldFromAlertToEvent [98]: Copies a single field from an existing alert to a deduplicating event for the same alert.
- copyFromAlertToEvent [99]: Copies multiple fields from an existing alert to a deduplicating event for the alert.
- copyFromContext [100]: Copies a field from the `workflowContext` to a destination object field. Sweep up filter applies.
- copyFromEventToAlert [101]: Allows you to copy `custom_info` fields from an incoming event to the existing alert that the event would de-duplicate to.
- copyToContext [103]: Copies an object field to the `workflowContext`.
- copyToPayload [104]: Copies a value to the payload in `workflowContext` for the current object.
- createPayload [109]: Creates a `workflowContext` payload from the triggering object using a predefined payload map.
- dejaVu [114]: Allows you to determine if a piece of data has been seen previously.
- deleteEnrichment [116]:Removes data from the enrichment datastore.
- deltaEvent [121]: 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 [122]: 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 [124]: Allows you to prevent further processing of an event.
- estimateSeverity [125]: Uses a predefined classification algorithm to estimate event or alert severity. Sweep up filter applies.

- existingAlertFilter [126]: Returns `true` if the existing alert for a deduplicating event matches a SQL-like filter.
- getIntegrationConfig [137]: Retrieves an integration configuration and stores it in the `workflowContext` for subsequent actions to use.
- getPayload [144]: Creates a `workflowContext` payload from the triggering object from a predefined payload map. Sweep up filter applies.
- isClear [161]: Returns `true` if the object's severity level is Clear (0).
- isInSubnet [162]: Returns `true` when an IP address is present within a specified subnet. Sweep up filter applies.
- isNewerThan [163]: Returns `true` when the object age in seconds is less than a specified age in seconds. Sweep up filter applies.
- isNotClear [165]: Returns `true` if the object's severity level is not "Clear". Sweep up filter applies.
- isNotNull [166]: 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 [167]: 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 [168]: Returns `true` when the object age in seconds is older than a specified age in seconds. Sweep up filter applies.
- listContains [171]: Returns `true` when the array field you query contains some of your specified values. Sweep up filter applies.
- listContainsAll [172]: Returns `true` when the array field you query contains all of your specified values. Sweep up filter applies.
- listDoesNotContain [173]: Returns `true` when the array field you query contains none of your specified values. Sweep up filter applies.
- logCEvent [176]: 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 [177]: Logs a message to the Moogfarmd log.
- logWorkflowContext [178]: Logs the contents of `workflowContext` to the current Moogfarmd log file at a warning level.
- logWorkflowDuration [179]: Logs debug messages for the workflow execution duration.
- lowerCase [182]: Changes the value of a field to lower case. Sweep up filter applies.
- populateNamedTopology [184]: 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 [185]: Prepends a concatenated set of fields to an existing field, using a separator character.
- prependString [186]: Prepends a string to an existing field, using a separator character.
- removeItemsFromList [189]: Removes a set of specified items from an existing list and write the resulting list back to the source field, or optionally to a different destination field.
- restAsyncPost [196]: Makes a HTTP POST request with a JSON payload to a named REST endpoint.
- searchAndReplace [198]: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Sweep up filter applies.
- searchAndReplaceOrdered [200]: 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 [228]; Sends the in-scope object to a named workflow in an Inform based workflow engine.
- setAgent [230]: Sets the agent of the event or alert.
- setAgentLocation [231]: Sets the agent location of the event or alert.
- setAgentTime [232]: 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 [240]: Updates a single record in the enrichment datastore with data from an alert.

- setEnrichmentBulk [241]: Updates multiple records in the enrichment datastore with an array of data from an alert.
- setExternalId [242]: Sets the external ID of the event or alert.
- setManager [243] Sets the manager of the event or alerts.
- setSource [251] Sets the source of the event or alert.
- setSourceId [252] Sets the source ID of the event or alert.
- setCoreEventField [236]: Sets a single core event field to a value.
- simpleLookup [256]: Defines the lookup as two arrays of equal length. Sweep up filter applies.
- skip [259]: 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 [260]: 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 [261]: Splits a fully qualified domain name (FQDN) into a hostname/short name and a domain name and updates fields with the values.
- upperCase [265]: Changes the value of a field to uppercase. Sweep up filter applies.
- willCreateNewAlert [266]: Returns `true` if the event will create a new alert.
- willDeduplicateAlert [267]: Returns `true` if the event will deduplicate into an existing alert.
- workflowContextSearchAndReplace [268]: Works only on fields within the workflowContext. The extracted fields are copied into a fixed workflowContext location: `workflowContext.extract`.

Alert and enrichment functions

The following functions are available in alert and enrichment workflows:

- ackNotification [49]: Automatically acknowledges a notification for a service.
- activateTopology [50]: Updates a named topology from an inactive to an active state.
- addDefaultCustomInfoValues [52]: Supersedes the existing `addDefaultValues` which used the now deprecated **Payload Maps** integration for the values.
- addDefaultValues [55]: Adds a set of default values to `custom_info` based on a payload map. Sweep up filter applies.
- addItemToList [58]: Adds an item or items to an array. Sweep up filter applies.
- addLocationData [56]: Populates the standard address fields in the standard `custom_info.location` object.
- addRestHeader [59]: Adds additional headers to an outbound request in a exportViaRest or sendViaRest workflow.
- addTags [60]: Adds or updates a custom info field called "tags" with an array of string values.
- addToContext [62]: Updates the workflow context with a key: value pair.
- addTopologyLink [64]: Creates a link between two endpoints, A (source node) and Z (sink node), in a named topology.
- addTopologyNode [65]: Creates a node in a named topology.
- alertDelta [66]: Returns `true` when attributes have changed.
- alertInSituation [67]: Returns `true` when the alert is a member of an active Situation. Sweep up filter applies.
- alertNotInSituation [68]: Returns `true` when the alert is not a member of an active Situation. Sweep up filter applies.
- appendFields [69]: Appends a concatenated set of fields to an existing field, using a separator character.
- appendString [70]: Appends a static string to an existing field separated by a space character.
- assignAlert [71]: Assigns an owner of in-scope alerts. Sweep up filter applies.
- assignAndAcknowledge [72]: Assigns and acknowledges the specified user as the owner of the alerts or Situations in scope.
- basicMaths [74]: Allows basic maths (+, -, * , / , %) to be performed on two fields that write the result to a destination field in either `custom_info` or the workflow context.
- between [76]: Returns `true` if the object creation date falls between two times.
- ceventFilter [77]: Returns `true` if the object matches a SQL-like filter. Sweep up filter applies.
- checkSeverity [78]: Checks the severity level of the object.
- checkTopology [81]: Checks for the existence of a named topology.
- checkTopologyLink [82]: Checks for a link between two endpoints, A (source node) and Z (sink node), in a named topology.
- cloneTopology [84]: Copies an existing topology to a new inactive named topology if the name is not already in use.
- closeAlert [85]: Closes alerts.
- compareFields [88]: Returns true or false based on the comparison of two string or number values.
- concatFields [89]: Sets the value of a field to a string representing a set of concatenated fields.
- contextFilter [91]: Filters a `workflowContext` object for a specified name field. Sweep up filter applies.
- convertField [92]: Converts a field using mappings defined in the Conversion Maps integration tile.
- convertStringToList [96]: Converts a string of words separated by a separator character into a Javascript array (list).
- convertToJSON [97]: Converts the object to JSON and adds it to the workflowContext for use in subsequent actions.
- copyFromContext [100]: Copies a field from the `workflowContext` to a destination object field. Sweep up filter applies.

- copyToContext [103]: Copies an object field to the `workflowContext`.
- copyToPayload [104]: Copies a value to the payload in `workflowContext` for the current object.
- createNotification [108]: Automatically creates a notification for a service.
- createPayload [109]: Creates a `workflowContext` payload from the triggering object using a predefined payload map.
- createTopology [111]: Creates a named topology if it does not already exist. Takes no action if the topology exists.
- deactivateTopology [112]: Updates a named topology from an active to an inactive state.
- deassignAlert [113]: Removes the current owner of in-scope alerts. Sweep up filter applies.
- dejaVu [114]: Allows you to determine if a piece of data has been seen previously.
- deleteEnrichment [116]:Removes data from the enrichment datastore.
- deleteTopology [118]: Delete a named topology
- deleteTopologyLink [119]: Removes a direct link between two endpoints, A (source node) and Z (sink node), in a named topology.
- deleteTopologyNode [120]: Deletes a node in a named topology.
- dnsLookup [122]: 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 [123]: Returns `true` when the in-cope alert or Situation is not in any of the specified states.
- estimateSeverity [125]: Uses a predefined classification algorithm to estimate event or alert severity. Sweep up filter applies.
- exportViaKafka [127]: Exports the payload from a createPayload [109] to an external Kafka endpoint. Sweep up filter applies.
- exportViaRest [129]: Exports the payload from a createPayload [109] to an external REST endpoint. Sweep up filter applies.
- forward [133]: Forwards the object to the named Moolet.
- getEnrichment [135]: Retrieves data from the enrichment datastore through the Cisco Crosswork Situation Manager Enrichment API. Sweep up filter applies.
- getPayload [144]: Creates a `workflowContext` payload from the triggering object from a predefined payload map. Sweep up filter applies.
- getIntegrationConfig [137]: Retrieves an integration configuration and stores it in the `workflowContext` for subsequent actions to use.
- hasNotSweptUp [153]: Returns true if the workflow entry filter did not sweep up any alerts or Situations.
- hasStatus [156]: Returns `true` when the in-scope alert or Situation is in any of the specified states.
- hasSweptUp [157]: Returns true if the workflow entry filter swept up any alerts or Situations.
- isAssigned [160]: Returns `true` if the object has an owner or moderator. Sweep up filter applies.
- isClear [161]: Returns `true` if the object's severity level is Clear (0).
- isInSubnet [162]: Returns `true` when an IP address is present within a specified subnet. Sweep up filter applies.
- isNewerThan [163]: Returns `true` when the object age in seconds is less than a specified age in seconds. Sweep up filter applies.
- isNotAssigned [164]: Returns `true` if the object does not have an owner or moderator. Sweep up filter applies.
- hasSweptUp [157]: Returns true if the workflow entry filter swept up any alerts or Situations.
- isNotClear [165]: Returns `true` if the object's severity level is not "Clear". Sweep up filter applies.
- isNotNull [166]: 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 [167]: 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 [168]: Returns `true` when the object age in seconds is older than a specified age in seconds. Sweep up filter applies.

- listContains [171]: Returns `true` when the array field you query contains some of your specified values. Sweep up filter applies.
- listContainsAll [172]: Returns `true` when the array field you query contains all of your specified values. Sweep up filter applies.
- listDoesNotContain [173]: Returns `true` when the array field you query contains none of your specified values. Sweep up filter applies.
- logCEvent [176]: 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 [177]: Logs a message to the Moogfarmd log.
- logWorkflowContext [178]: Logs the contents of `workflowContext` to the current Moogfarmd log file at a warning level.
- logWorkflowDuration [179]: Logs debug messages for the workflow execution duration.
- lookupAndReplace [180]: 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 [182]: Changes the value of a field to lower case. Sweep up filter applies.
- notifySlack [183]: Sends a request to Slack channel(s).
- populateNamedTopology [184]: 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 [185]: Prepends a concatenated set of fields to an existing field, using a separator character.
- prependString [186]: Prepends a string to an existing field, using a separator character.
- removeItemsFromList [189]: Removes a set of specified items from an existing list and write the resulting list back to the source field, or optionally to a different destination field.
- replaceString [192]: Replaces a string or regular expression in a field with a specified string or regular expression.
- resolveNotification [194]: Automatically resolves a notification for a service.
- restAsyncPost [196]: Makes a HTTP POST request with a JSON payload to a named REST endpoint.
- searchAndReplace [198]: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Sweep up filter applies.
- searchAndReplaceOrdered [200]: 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.
- sendEmail [208]: Specifies the email server instance defined in the integrations UI, the email address(s), email subject, and email message that will be used for the email request.
- sendEmailUsingTemplate [212]: Specifies the email server instance and message template defined in the integrations UI, and the email address(s) that will be used for an email request.
- sendMooletInform [215]: Sends a Moolet inform with a subject and details.
- sendSituationsToWorkflow [218]: Sends the member alert of active Situations to a named workflow within a named Inform based workflow engine using the moogdb sendToWorkflow API call.
- sendToWorkflow [228]; Sends the in-scope object to a named workflow in an Inform based workflow engine.
- sendViaRest [229]: Sends the payload from a createPayload [109] to an external REST endpoint. Sweep up filter applies.
- setAgent [230]: Sets the agent of the event or alert.
- setAgentLocation [231]: Sets the agent location of the event or alert.
- setClass [235]: Sets the class of the alert.
- setCustomInfoJSONValue [237]: Adds or updates a custom info key to the specified JSON value. Sweep up filter applies.
- setCustomInfoValue [238]: Adds or updates a custom info key to a specified string value. Sweep up filter applies.
- setDescription [239]: Sets the description of the object.
- setEnrichment [240]: Updates a single record in the enrichment datastore with data from an alert.

- setEnrichmentBulk [241]: Updates multiple records in the enrichment datastore with an array of data from an alert.
- setExternalId [242]: Sets the external ID of the event or alert.
- setManager [243] Sets the manager of the event or alerts.
- setSlackTarget [250]: Sets the target Slack channel and title for a message payload.
- setSource [251] Sets the source of the event or alert.
- setSourceId [252] Sets the source ID of the event or alert.
- setSeverity [245]: Sets the severity of the alert. Sweep up filter applies.
- setType [253]: Sets the type of the alert.
- simpleLookup [256]: Defines the lookup as two arrays of equal length. Sweep up filter applies.
- skip [259]: 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 [260]: 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 [261]: Splits a fully qualified domain name (FQDN) into a hostname/short name and a domain name and updates fields with the values.
- upperCase [265]: Changes the value of a field to uppercase. Sweep up filter applies.
- workflowContextSearchAndReplace [268]: Works only on fields within the workflowContext. The extracted fields are copied into a fixed workflowContext location: `workflowContext.extract`.

Situation functions

The following functions are available in Situation workflows:

- ackNotification [49]: Automatically acknowledges a notification for a service.
- addCorrelationInfo [51]: Adds an External ID to a Situation.
- addDefaultCustomInfoValues [52]: Supersedes the existing `addDefaultValues` which used the now deprecated **Payload Maps** integration for the values.
- addDefaultValues [55]: Adds a set of default values to `custom_info` based on a payload map. Sweep up filter applies.
- addItemToList [58]: Adds an item or items to an array. Sweep up filter applies.
- addLocationData [56]: Populates the standard address fields in the standard `custom_info.location` object.
- addRestHeader [59]: Adds additional headers to an outbound request in a exportViaRest or sendViaRest workflow.
- addTags [60]: Adds or updates a custom info field called "tags" with an array of string values.
- addThreadEntry [61]: Adds a post to the named thread in the Collaboration tab of the Situation Room.
- addToContext [62]: Updates the workflow context with a key: value pair.
- appendFields [69]: Appends a concatenated set of fields to an existing field, using a separator character.
- appendString [70]: Appends a static string to an existing field separated by a space character.
- assignAndAcknowledge [72]: Assigns and acknowledges the specified user as the owner of the alerts or Situations in scope.
- assignModerator [73]: Assigns a user as the moderator of the Situations in scope.
- basicMaths [74]: Allows basic maths (+, -, * , / , %) to be performed on two fields that write the result to a destination field in either `custom_info` or the workflow context.
- between [76]: Returns `true` if the object creation date falls between two times.
- ceventFilter [77]: Returns `true` if the object matches a SQL-like filter. Sweep up filter applies.
- checkSeverity [78]: Checks the severity level of the object.
- checkSituationFlag [79]: Checks if a specific flag is set for a Situation.
- checkSituationState [80]: Returns true if the specified state exists for a Situation. Sweep up filter applies.
- compareFields [88]: Returns true or false based on the comparison of two string or number values.
- concatFields [89]: Sets the value of a field to a string representing a set of concatenated fields.
- containsAlertDetails [90]: Returns `true` if all or any of the alerts in the Situation matches the filter condition. Sweep up filter applies.
- contextFilter [91]: Filters a `workflowContext` object for a specified name field. Sweep up filter applies.
- convertField [92]: Converts a field using mappings defined in the Conversion Maps integration tile.
- convertStringToList [96]: Converts a string of words separated by a separator character into a Javascript array (list).
- convertToJSON [97]: Converts the object to JSON and adds it to the workflowContext for use in subsequent actions.
- copyFromContext [100]: Copies a field from the `workflowContext` to a destination object field. Sweep up filter applies.
- copyToContext [103]: Copies an object field to the `workflowContext`.
- copyToPayload [104]: Copies a value to the payload in `workflowContext` for the current object.
- createNotification [108]: Automatically creates a notification for a service.
- createPayload [109]: Creates a `workflowContext` payload from the triggering object using a predefined payload map.
- dnsLookup [122]: 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 [127]: Exports the payload from a createPayload [109] to an external Kafka endpoint. Sweep up filter applies.
- exportViaRest [129]: Exports the payload from a createPayload [109] to an external REST endpoint. Sweep up filter applies.
- createServiceTicket [110]: Creates a ticket for the specified service.
- dejaVu [114]: Allows you to determine if a piece of data has been seen previously.
- doesNotHaveStatus [123]: Returns `true` when the in-cope alert or Situation is not in any of the specified states.
- filterByCookbook [130]: Returns `true` if the Visualize data for the Situation matches the cookbook name.
- filterByCookbookAndRecipe [131]: Returns `true` if the Visualize data for the Situation matches the cookbook name and recipe name.
- filterByRecipe [132]: Returns `true` if the Visualize data for the Situation matches the recipe name.
- forward [133]: Forwards the object to the named Moolet.
- getCorrelationInfo [134]: Checks whether correlation info exists for a Situation.
- getIntegrationConfig [137]: Retrieves an integration configuration and stores it in the `workflowContext` for subsequent actions to use.
- getPayload [144]: Creates a `workflowContext` payload from the triggering object from a predefined payload map. Sweep up filter applies.
- getSituationFlags [145]: Retrieves the Situation flags and stores them in the workflowContext for subsequent actions to use.
- getThreadEntry [148]: Returns a thread entry for a thread in a Situation.
- getVisualizationData [149]: Retrieves the Visualize data and stores them in the workflowContext for subsequent actions to use.
- hasCausalPRC [150]: Returns `true` if one or more alerts in the Situation has a causal PRC flag set. Sweep up filter applies.
- hasMerged [151]: Returns `true` if the Situation has been merged or superseded.
- hasNotMerged [152]: Returns `true` if the Situation has not been merged or superseded.
- hasNotSweptUp [153]: Returns true if the workflow entry filter did not sweep up any alerts or Situations.
- hasPRCAlert [154]: Returns true if a situation has at least one alert with a PRC value set (an rc_probability).
- hasSimilarSituations [155]: Returns `true` when the Situation has a similar Situation above the specified threshold.
- hasStatus [156]: Returns `true` when the in-scope alert or Situation is in any of the specified states.
- isAlertAcknowledged [158]: Returns `true` when the in-scope alert state is Acknowledged.
- isAlertNotAcknowledged [159]: Returns `true` when the in-scope alert state is not Acknowledged.
- isAssigned [160]: Returns `true` if the object has an owner or moderator. Sweep up filter applies.
- isClear [161]: Returns `true` if the object's severity level is Clear (0).
- isNotAssigned [164]: Returns `true` if the object does not have an owner or moderator. Sweep up filter applies.
- isNewerThan [163]: Returns `true` when the object age in seconds is less than a specified age in seconds. Sweep up filter applies.
- isNotClear [165]: Returns `true` if the object's severity level is not "Clear". Sweep up filter applies.
- isNotNull [166]: 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 [167]: 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 [168]: Returns `true` when the object age in seconds is older than a specified age in seconds. Sweep up filter applies.
- isPrimaryTeamSet [169]: Returns true if the primary team for a Situation is set.
- labelSituation [170]: Labels the Situation using the Situation Manager Labeler macro language. Sweep up filter applies.

- listContains [171]: Returns `true` when the array field you query contains some of your specified values. Sweep up filter applies.
- listContainsAll [172]: Returns `true` when the array field you query contains all of your specified values. Sweep up filter applies.
- listDoesNotContain [173]: Returns `true` when the array field you query contains none of your specified values. Sweep up filter applies.
- listSituationAlertIds [174]: Adds the current alert Ids in the Situation into the workflow context under `situationAlertIds`.
- listSituationHosts [175]: Adds the current host names in the Situation into the workflow context under `situationHosts`.
- logCEvent [176]: 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 [177]: Logs a message to the Moogfarmd log.
- logWorkflowContext [178]: Logs the contents of `workflowContext` to the current Moogfarmd log file at a warning level.
- logWorkflowDuration [179]: Logs debug messages for the workflow execution duration.
- lowerCase [182]: Changes the value of a field to lower case. Sweep up filter applies.
- notifySlack [183]: Sends a request to Slack channel(s).
- prependFields [185]: Prepends a concatenated set of fields to an existing field, using a separator character.
- prependString [186]: Prepends a string to an existing field, using a separator character.
- removeItemsFromList [189]: Removes a set of specified items from an existing list and write the resulting list back to the source field, or optionally to a different destination field.
- removeSituationFlag [191]: Removes a specific flag from a Situation.
- replaceString [192]: Replaces a string or regular expression in a field with a specified string or regular expression.
- resolveNotification [194]: Automatically resolves a notification for a service.
- resolveSituation [195]: Marks in-scope Situations as Resolved if they match the workflow's entry filter and sweep up filter.
- reviveSituation [197]: Revives (sets to Open) a Situation that is currently set to Resolved.
- restAsyncPost [196]: Makes a HTTP POST request with a JSON payload to a named REST endpoint.
- searchAndReplace [198]: Matches a regular expression to an object field and maps the contents of subgroups to other fields. Sweep up filter applies.
- searchAndReplaceOrdered [200]: 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.
- sendAlertsToWorkflow [205]: Sends Situation alerts to a named workflow within a named Inform based workflow engine using the moogdb sendToWorkflow API call.
- sendEmail [208]: Specifies the email server instance defined in the integrations UI, the email address(s), email subject, and email message that will be used for the email request.
- sendEmailUsingTemplate [212]: Specifies the email server instance and message template defined in the integrations UI, and the email address(s) that will be used for an email request.
- sendMooletInform [215]: Sends a Moolet inform with a subject and details.
- sendToWorkflow [228]; Sends the in-scope object to a named workflow in an Inform based workflow engine.
- sendViaRest [229]: Sends the payload from a createPayload [109] to an external REST endpoint. Sweep up filter applies.
- setCustomInfoJSONValue [237]: Adds or updates a custom info key to the specified JSON value. Sweep up filter applies.
- setCustomInfoValue [238]: Adds or updates a custom info key to a specified string value. Sweep up filter applies.
- setDescription [239]: Sets the description of the object.

- setSituationFlag [246]: Sets a flag for a Situation.
- setSituationServices [247]: Sets the impacted services for a Situation.
- setSlackTarget [250]: Sets the target Slack channel and title for a message payload.
- sigActionFilter [254]: Returns `true` if the Situation action is of the specified type.
- sigActionToolFilter [255]: Returns `true` if the specified tool has been run against a Situation.
- simpleLookup [256]: Defines the lookup as two arrays of equal length. Sweep up filter applies.
- situationDelta [258] Returns `true` when attributes have changed.
- skip [259]: 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 [260]: 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 [265]: Changes the value of a field to uppercase. Sweep up filter applies.
- workflowContextSearchAndReplace [268]: Works only on fields within the workflowContext. The extracted fields are copied into a fixed workflowContext location: `workflowContext.extract`.

Infrastructure and Automation functions

The following functions are available in specific infrastructure and automation workflows:

- getJDBCEnrichment [142]: Adds data to alerts from a JDBC database. Available in JDBC Enrichment workflows.
- getServiceNowEnrichment [146]: Adds data to alerts from a ServiceNow database.
- processMicroFocusOOAutomationResponse [187]: Processes the response from Micro Focus Operations Orchestration automation.
- sendMicroFocusOOAutomationRequest [214]: Sends a request to Micro Focus Operations Orchestration to launch a flow.
- sendRequestToAutomation [217]: Sends a request to the corresponding outbound automation workflow to invoke specified action for the named service.
- sendToAnsible [225]: Sends an automation request to Ansible. Available in Ansible Alert and Ansible Situation workflows.
- sendToAutomation [226]: Sends an automation request. Available in EyeShare Alert, EyeShare Situation, Ignio Alert, and Ignio Situation workflows.
- sendToPuppet [227]: Sends an automation request to Puppet. Available in Puppet Alert and Puppet Situation workflows.
- setAnsibleJob [233]: Sets the instance and job template rule to use for Ansible automation requests. Available in Ansible Alert and Ansible Situation workflows.
- setAutomationPayload [234]: 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 [244]: Sets the instance and job template rule to use for Puppet automation requests. Available in Puppet Alert and Puppet Situation workflows.

Ticketing integration functions

The following functions are available in specific ticketing workflows:

- closeAlertOpsIncident [86]: Resolves the corresponding AlertOps alert.
- closeIncident [87]: Sends a request to the corresponding outbound integration workflow to close an incident for the named service.
- createAlertOpsIncident [106]: Creates a new AlertOps alert for a Situation.
- createIncident [107]: Sends a request to the corresponding outbound integration workflow to create a new incident for the named service.
- receiveUpdateFromAlertOpsIncident [188]: Updates Situations with responses from AlertOps.
- resolveIncident [193]: Sends a request to the corresponding outbound integration workflow to resolve an incident for the named service.
- sendAcknowledgedToIncident [204]: Sends a request to the corresponding outbound integration workflow to acknowledge an incident for the named service.
- sendAssignedToAlertOpsIncident [206]: Assigns the corresponding AlertOps alert to a user corresponding to the Situation moderator.
- sendAssignedToIncident [210]: Sends a request to the corresponding outbound integration workflow to assign the incident for the named service.
- sendCIsAddedToIncident [207]: Sends a request to the corresponding outbound integration workflow to add a list of CIs to the incident for the named service.
- sendNoteToIncident [216]: Sends a request to the corresponding outbound integration workflow to post a message to the incident for the named service.
- sendTeamsAddedToAlertOpsIncident [219]: Adds "recipients" to the corresponding AlertOps alert for teams added to a Situation.
- sendTeamsAddedToIncident [220]: Sends a request to the corresponding outbound integration workflow to add a list of teams to the incident for the named service.
- sendTeamsRemovedToIncident [222]: Sends a request to the corresponding outbound integration workflow to remove a list of teams from the incident for the named service.
- sendThreadEntryToAlertOpsIncident [223]: Sends a reply to the corresponding AlertOps alert.
- sendThreadEntryToIncident [224]: Sends a request to the corresponding outbound integration workflow to update the incident for the named service using thread entry retrieved by a preceding `getThreadEntry` action.
- updateAlertOpsIncident [263]: Sends generic updates to the AlertOps alert.
- updateIncident [264] Sends a request to the corresponding outbound integration workflow to perform a generic update to an incident for the named service.

ackNotification

A Workflow Engine function that automatically acknowledges a notification for a service.

This function currently supports the PagerDuty and Opsgenie 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 [108] 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 [35].

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 [35].

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"
}
```

addCorrelationInfo

A Workflow Engine function that adds an External ID to a Situation.

You can use this function to store external ticket references against a Situation. `addCorrelationInfo` acts as a wrapper around the method `moogdb.addSigCorrelationInfo`.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `addCorrelationInfo` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	Name of the external service, such as `'ServiceNow'`
`externalId`	yes	string	Identifier that the entity has in the external service, such as `'incident reference'`

Example

The following example demonstrates typical use of Workflow Engine function `addCorrelationInfo`.

You have a ticketing integration that triggers a request to an external ticketing application to create a ticket based on the contents of a Situation. You can store the external ticket reference that the external application returns against the originating situation. To add the ticket reference with the `workflowContext` key `myTicketID` to a Situation that is using the external service `myTicketApp`, set the following:

- `serviceName`: myTicketApp
- `externalId`: workflowContext.myTicketID

The UI translates your settings to the following JSON:

```
{ “serviceName”:”myTicketApp”, "externalId": "workflowContext.myTicketID" }
```

You may receive a ticket reference from the ticketing application synchronously (at request time) or asynchronously via a callback.

If you receive the ticket reference synchronously, then a separate action initiates the requests to the external ticketing application and returns the ticketing reference to the `workflowContext` before the `addCorrelationInfo` action.

If you receive the the ticket reference asynchronously via callback, you trigger a response to a Situation Inform Workflow by adding a hook to the external application and use the `sendToWorkflow` Graze endpoint to trigger a response that contains the `addCorrelationInfo` action.

addDefaultCustomInfoValues

A Workflow Engine action that uses the Payloads integration to define a set of default values to add to an event, alert or Situation `custom_info`. The contents of the payload can use the full Payload macros in the definition, including fixed and substituted values. This makes it easier to move existing object values from a non-standard `custom_info` schema to a fixed schema.

This action supersedes the existing `addDefaultValues` which used the deprecated Payload Maps integration for the values.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for event, alert, and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `addDefaultCustomInfoValues` takes the following arguments:

Name	Required	Type	Description
mapName	yes	string	The Payload map name defined in the Payloads integration tile.
key	yes	string	The destination key under `custom_info` key where you want to store the data.

Example

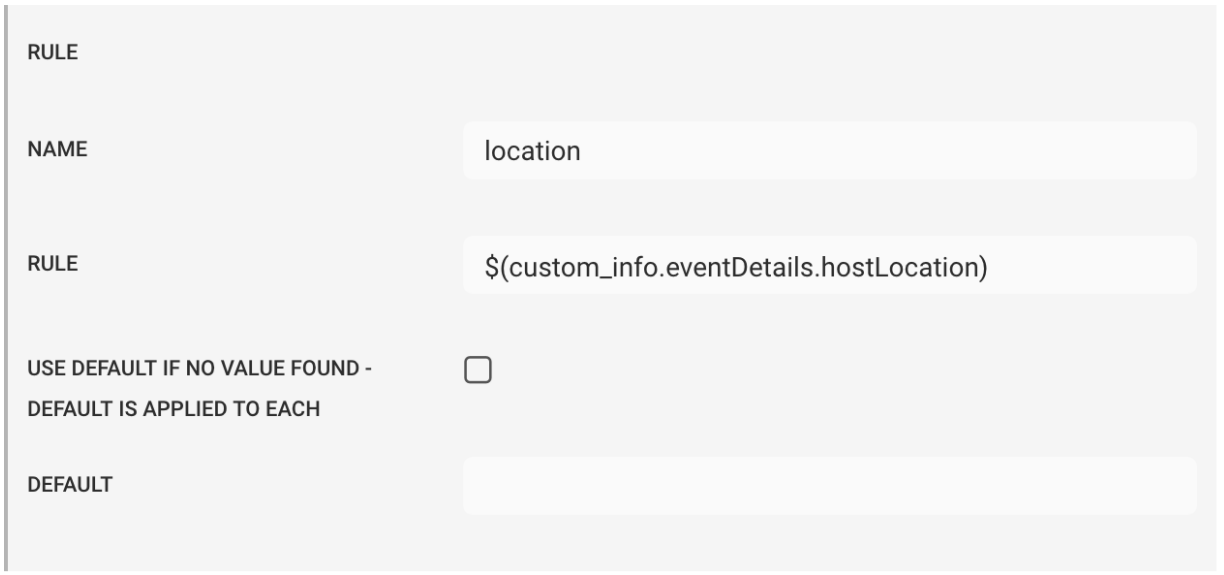
An event has a fixed schema containing the following `custom_info`:

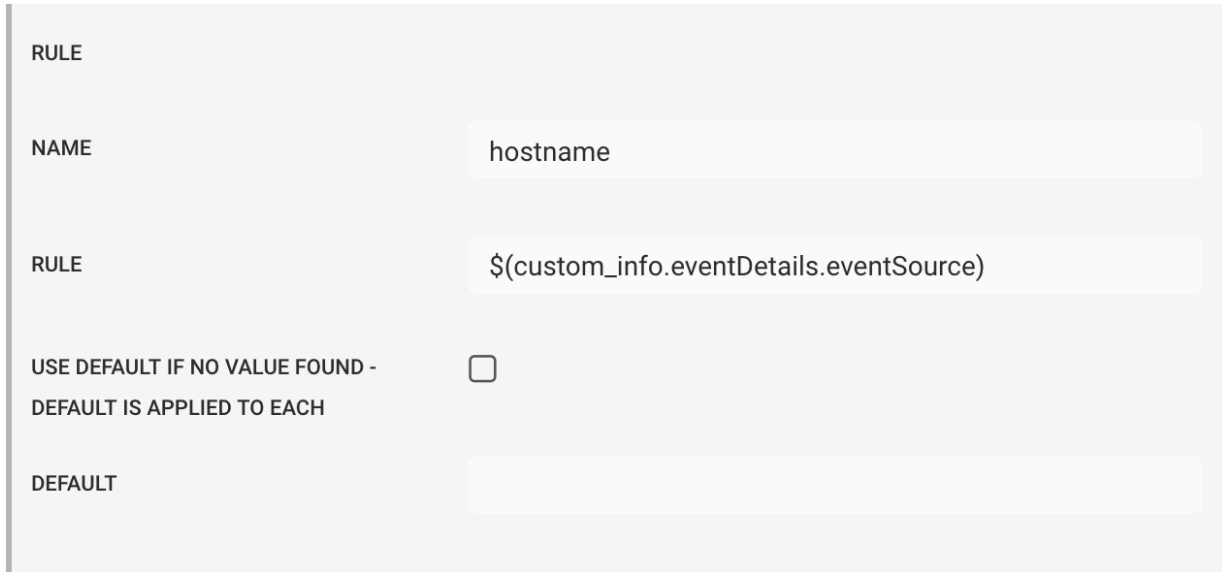
```
"custom_info" {
  "eventDetails" : {
    "eventSource" : "host123",
    "hostLocation" : "London"
    }
}
```

If you wanted to create a populated `custom_info` model including an empty list for “applications”:

```
"custom_info" {
  "eventAttributes" {
    "location" : ...,,
    "hostname" : ....,
    "applications" : []
  },
  "eventDetails" : {
    ...
  }
}
```

You can define a Payload containing the following rules in the Payload and the `addDefaultCustomInfoValues` in an event workflow.:





And the following arguments in the action:

- **mapName** : defaults
- **key** : `custom_info.eventAttributes`

The UI translates your settings to the following JSON:

```
{"mapName":"defaults","key":"custom_info.eventAttributes"}
```

Before running the event's `custom_info` would be:

```
"custom_info": {
        "eventDetails": {
            "eventSource": "host123",
            "hostLocation": "London"
        }
    }
```

After running it would be:

```
"custom_info": {
        "eventDetails": {
            "eventSource": "host123",
            "hostLocation": "London"
        },
        "eventAttributes": {
            "location": "London",
            "hostname": "host123",
            "applications": []
        }
    }
```

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 [35].

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.

addLocationData

A Workflow Engine function that populates the standard address fields in the standard `custom_info.location` object. All of the parameters are optional, and all allow substitution. For this reason, the "number" field is treated as a string and also allows names instead of numbers.

The destination for this data is hard-coded to the known `custom_info.location` object. You can copy the data to other fields using workflow actions, but the initial destination is fixed. You can omit details, and multiple runs (example: via an external `sendToWorkflow` function) updates the existing items only and not remove/blank out previously set details.

There is no validation of the contents of the values (for example, the country is not checked for a real country). Use any values may be used as needed.

This function is available as a feature of the Add-ons v2.2 download and later.

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 [35].

Arguments

Workflow Engine function `addLocationData` takes the following arguments:

Name	Required	Type	Description
`number`	no	string	The street number, name, etc.
`street`	no	string	The street name.
`city`	no	string	The city name.
`state`	no	string	The state or equivalent name (such as country).
`region`	no	string	The region name.
`country`	no	string	The country name.
`zipCode`	no	string	The ZIP or postal code.

Example

The following example demonstrates typical use of Workflow Engine `addLocationData` function.

To set a basic address for an alert in an Alert Workflow:

- `number` : 223
- `street` : Main St.
- `city` : BigTown
- `state` : New York
- `region` : Eastern US
- `country` : USA
- `zipCode` : 123456

The UI translates your settings to the following JSON:

```
{"number":223,"street":"Main St.","city":"BigTown","state":"New
York","region":"Eastern US","country":"USA","zipCode":"123456"}
```

The function can use substitution if needed, from either the existing object or the workflowContext as needed.

Example:

```
{"street":"$(workflowContext.street)","city":"$(workflowContext.city)"}
```

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 [35].

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.

addRestHeader

A Workflow Engine function that adds additional headers to an outbound request in a exportViaRest or sendViaRest workflow. This workflow must run before the exportViaRest or sendViaRest action to be effective.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `addRestHeader` takes the following arguments:

Name	Required	Type	Description
`name`	yes	string	The header name.
`value`	yes	string	The header value.

Example

The following example demonstrates typical use of Workflow Engine function `addRestHeader`.

To add an header called “API key” with a value “1234567890ABCDEFGH”, set the following:

- `name : “API key”`
- `value : “1234567890ABCDEFGH”`

The UI translates your settings to the following JSON:

```
{"name":"API key","value":"1234567890ABCDEFGH"}
```

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 [35].

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 [35].

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"}
```

addToContext

A Workflow Engine function that adds a specified value to a specified key in the workflow context. The key can be nested (example: key1.key2) and the value can be any acceptable value, including full substitution using the macro language. Subsequent actions place the key and value into the workflow context for use.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for event, alert, and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `addToContext` takes the following arguments:

Name	Required	Type	Description
`key`	yes	string	The key name. Compound keys are added as nested objects.
`value`	yes	string	The value to set the key to. The value can include native values (strings, numbers) or substituted values.

Example

The following example demonstrates typical use of Workflow Engine function `addToContext`.

To add a simple value to the workflowContext

- `key` : myKey
- `value` : A simple value, 1, 2, 3, 4

The UI translates your settings to the following JSON:

```
{"key":"myKey","value":"A simple value 1, 2, 3, 4"}
```

To add a JSON object (in this case a list, or array), use the macro language:

- `key` : myKey1
- `value` : $TO_JSON([1, 2, 3, 4, 5,])

The UI shows this as:

```
{"key":"myKey2","value":"$TO_JSON([1,2,3,4,5])"}
```

To add a substituted value from the in-scope object (in this case a Situation):

- `key` : myKey3
- `value` : A substitution $(sig_id) with an expansion $EXPAND(internal_priority)

The UI shows this as:

```
{"key":"myKey3","value":"A substitution $(sig_id) with an expansion
$EXPAND(internal_priority) "}
```

To add a nested object, use a dotted notation in the key name:

- `key` : myKey4.myKey5
- `value` : $TO_JSON({ "a" : "b" , "c" : [1, 2 ,3, 4, 5] })

The UI shows this as:

```
{"key":"myKey4.myKey5","value":"$TO_JSON({ \"a\" : \"b\" , \"c\" : [ 1,
2 ,3, 4, 5] })"}
```

After the above actions, the workflowContext looks like this:

```
WORKFLOW CONTEXT: SITUATION: 186 :  :
{
    "myKey": "A simple value 1, 2, 3, 4",
    "myKey2": [
        1,
        2,
        3,
        4,
        5
    ],
    "myKey3": "A substitution 186 with an expansion Critical ",
    "myKey4": {
        "myKey5": {
            "a": "b",
            "c": [
                1,
                2,
                3,
                4,
                5
            ]
        }
    }
```

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 [35].

Arguments

Workflow Engine function `addTopologyLink` takes the following arguments:

Name	Required	Type	Description
topology-Name	yes	string	The name or substited value for the topology. To substitute a value, use $(<attrib-ute_name>). For example $(custom_info.myTopology)
sourceNode	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a val-ue, 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 trig-gering 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:

- `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, 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 [35].

Arguments

Workflow Engine function `addTopologyNode` takes the following arguments:

Name	Required	Type	Description
topology-Name	yes	string	The name or substited value for the topology. To substitute a value, use $(<attrib-ute_name>). For example $(custom_info.myTopology).
nodeName	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a val-ue, 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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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`.

assignAndAcknowledge

A Workflow Engine function that assigns and acknowledges the specified user as the owner of the in-scope alerts or Situations. This includes the triggering alert or Situation and any associated alerts or Situations captured by a sweep up filter.

Specify users as follows:

- User Name
- User ID
- A workflow context key with a user name or user ID value.

To identify the user from a workflow context key, include `workflowContext` in the path. For example `workflowContext.userName`.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert and Situation workflows.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `assignAndAcknowledge` takes the following arguments:

Name	Required	Type	Description
user	yes	string	The User ID or User Name to assign as owner.

Example

The following example demonstrates typical use of Workflow Engine function `assignAndAcknowledge`. To assign and acknowledge with a specific user, set **user** field to the User ID. For example "support":

The UI translates the settings to the following JSON:

```
{"user":"support"}
```

The Workflow Engine assigns and acknowldeges all alerts or Situations in scope to the “support” user.

If you have populated the workflow context with the User Name or User ID, you can retrieve the owner from context. For example if the workflow context contains the following:

```
"userName" : "support"
```

You can specify the workflow context path in the **user** field as `workflowContext.userName`.

```
{"user":"workflowContext.userName"}
```

In this case, the Workflow Engine retrieves the username "support" from the workflow context and assigns and acknowledges the alerts and Situations accordingly.

If at least one alert or situation is assigned and acknowledged to a user, the action returns true. Otherwise it returns false.

assignModerator

A Workflow Engine function that ssigns a user as the moderator of the in-scope Situations. This means the triggering Situation and any associated situations captured by a sweep up filter.

Specify users as follows:

- User Name
- User ID
- A workflow context key with a user name or user ID value.

To identify the user from a workflow context key, include .`workflowContext` in the path. For example `workflowContext.userName`.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for Situation workflows only.

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `assignModerator` takes the following arguments:

Name	Required	Type	Description
user	yes	string	The User ID or User Name to assign as owner.

Example

The following example demonstrates typical use of Workflow Engine function `assignModerator`. To assign a specific user, set **user** field to the User ID. For example "support":

The UI translates the settings to the following JSON:

```
{"user":"support"}
```

The Workflow Engine assigns all in-scope Situations in scope to the moderator “support”.

If you have populated the workflow context with the User Name or User ID, you can retrieve the owner from context. For example if the workflow context contains the following:

```
"userName" : "support"
```

You can specify the workflow context path in the **user** field as `workflowContext.userName`.

```
{"user":"workflowContext.username"}
```

In this case, the Workflow Engine retrieves the username "support" from the workflow context and assigns and the Situations accordingly.

If at least one alert or situation is assigned and acknowledged to a user, the action returns true. Otherwise it returns false.

basicMaths

A Workflow Engine function for basic math: addtion(+), subtraction (-), multiplication (*), division (/), and percentage (%). You can perform math functions on two fields and write the result to a destination field in either `custom_info` or the workflow context. The action returns true when both sides of the operation are numbers and the evaluation results in a valid number.

Because the `basicMaths` action can write data into and read data from the workflowContext, you can use multiple actions to perform complex operations. For example, you can calculate the sum of two fields and evaluate the result as a percentage of the third field:

```
( ( field1 + field3 ) / field3 ) * 100
```

The `basicMaths` action would be spread across two actions:

Action 1

- Calculate the value of (field1 + field2).
- Write the result to the workflowContext (e.g. `workflowContext.step1Result`).

Action 2

- Calculate the percentage using (`workflowContext.step1Result` / field3) * 100.
- Write the result to a `custom_info` field or workflowContext for subsequent actions.

WARNING

The % operator in this action is NOT a modulus operator (remainder). It is used to calculate the percentage of the left hand side of the right hand side: ((left side / right side) * 100).

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for event, alert, and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `basicMaths` takes the following arguments:

Name	Required	Type	Description
leftSide	yes	string	An absolute value or substituted value for the left hand side of the operation.
operator	yes	string	One of `+` , `-` , `*` , `/` , `%`
rightSide	yes	string	An absolute value or substituted value for the righ hand side of the operation.
resultLocation	yes	string	Either a `custom_info` or workflowContext location to store the results.

Example

For this example, assume there is an alert enrichment that provides the total number of hosts in a specific location and you had a recipe based on that location. Also assume every alert in the situation was enriched with the same location and `location_hosts` number.

In the SituationMgLabeller you can extract this data and add it to the Situations `custom_info` using the $MAP[ ] macro:

```
$MAP[ $UCOUNT(source,affected_hosts)
$UNIQ(custom_info.enrichment.location_hosts)
$UNIQ(custom_info.enrichment.location_name, location)]
```

This add two values to the Situation custom_info:

```
"custom_info" : {
  "location" : "London",
  "affected_hosts" : 134,
  "location_hosts" : 200
}
```

In the Situation Workflow Engine you can use the `basicMaths` action to calculate the % affected in the location and add this to the `custom_info` (e,g. to allow this value to be used in re-labelling, or passed to an external system).

Set the following arguments:

- **leftSide** : $(custom_info.affected_hosts)
- **operator** : %
- **rightSide** : $(custom_info.location_hosts
- **resultLocation** : custom_info.percentAffected

The UI translates your settings to the following JSON:

```
{"leftSide":"$(custom_info.affected_hosts)","operator":"%","rightSide":"$
(custom_info.location_hosts)","resultLocation":"custom_info.percentAffected"
}
```

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 [35].

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. See Filter Search Data for more information on filters.

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 [35].

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 [35].

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 [35].

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 [246].

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 [79].

The workflow sweep up filter applies to this function.

Back to Workflow Engine Functions Reference [35].

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 [35].

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 [35].

Arguments

Workflow Engine function `checkTopologyLink` takes the following arguments:

Name	Required	Type	Description
topology-Name	yes	string	The name or substited value for the topology. To substitute a value, use $(<attrib-ute_name>). For example $(custom_info.myTopology)
sourceNode	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a val-ue, 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 [35].

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":"severity"}
```

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 [35].

Arguments

Workflow Engine function `cloneTopology` takes the following arguments:

Name	Required	Type	Description
topology-Name	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 [35].

Arguments

Workflow Engine function `closeAlert` has no arguments.

closeAlertOpsIncident

A Workflow Engine function that resolves the corresponding AlertOps alert.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `closeAlertOpsIncident` has no arguments.

Example

The following example demonstrates typical use of Workflow Engine function `closeAlertOpsIncident`.

This function, in conjunction with the AlertOps integration, sends a REST request to AlertOps to the corresponding alert.

The integration adds a new workflow “Close AlertOps Incident” to the “Situation Integration” Workflow Engine containing this action.

closeIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to close an incident for the named service.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `closeIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value alertops.
`instanceName`	no	string	An instance name for integrations supporting multiple instances; defaults to the first instance.
`serviceName`	no	string	A template name for integrations supporting predefined templates. If unspecified, a default template is used.
`serviceName`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

compareFields

A Workflow Engine function that returns true or false based on the comparison of two string or number values.

You can use this function as an action in a workflow. The JavaScript `localCompare()` method compares the two strings. Both the left-hand and right-hand side are converted to lower case before comparison takes place. Valid operators are: >, <, >=, <=, `!=`, =. The = operator translates to the JavaScript strict equality operator === and the `!=` operator translates to the strict inequality operator `!==`.

If the comparison returns `true`, then `compareFields` returns `true`. Otherwise, `compareFields` returns `false`.

If either the left-hand side or right-hand value of the comparison is `null`, then `compareFields` returns `false`.

Coercion takes place when one side of the comparison is a string of digits and the other side is a number. If coercion fails, then `compareFields` will return `false`. If the value for either side is not a number or a string, then `compareFields` will return `false`.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

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 [35].

Arguments

Workflow Engine function `compareFields` takes the following arguments:

Name	Required	Type	Description
`leftSide`	Yes	String	The left-hand side of the comparison expression. A condition that can be evaluated to return a `true` or `false` value. This argument can be substituted.
`operator`	Yes	String	A binary comparison operator that takes two operands and returns `true` or `false`. Do not user quotes around the operator.
`rightSide`	Yes	String	The right-hand side of the comparison expression. A condition that can be evaluated to return a `true` or `false` value. This argument can be substituted.

Example

The following example demonstrates typical use of Workflow Engine function `compareFields`.

You want to increase the severity of an alert to `critical` if the value of `custom_info.impact` is greater than or equal to 1000. Create an Alert Workflow with a `compareFields` action and a `setSeverity` action. Set the following for `compareFields`:

- `leftSide`: $TO_INT(custom_info.impact)
- `operator`: `>=`
- `rightSide`: 1000

The UI translates your settings to the following JSON:

```
{"leftSide":"$TO_INT(custom_info.impact)","operator":">=","rightSide":"1000"
}
```

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 [35].

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. See Filter Search Data for more information on filters.

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 [35].

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 [35].

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`.

convertField

A Workflow Engine function that converts a field value using mappings from the Conversion Maps integration.

The action looks up the value of the field in the conversion map. You can write the results of the lookup to a separate field with the optional **target** argument. If you don't specify a **target**, the function updates the source field in-line. Both the **field** and **target** arguments accept workflow context and payload keys. Specify the appropriate prefix, either `workflowContext` or `payload`.

The optional **reversed** argument defaults to “false”. If true, the lookup is performed in reverse. The Workflow Engine checks for the value against the conversion map `alias` to return the `name`.

The keys and values in the conversion maps are stored as strings. The optional **type** argument you to explicitly cast to one of the following primitive types: `number`, `string`, or `boolean`. For core target fields, there is implicit casting.

The return behavior of the action is controlled by the “no match” behavior settings in the conversion maps. See the examples below for details.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for event, alert, and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `convertField` takes the following arguments:

Name	Required	Type	Description
mapName	yes	string	Name of mapping.
field	yes	string	Name of field to convert.
target	no	string	Optional target field. If omitted, the supplied field will be updated.
type	no	string	Optional primitive return type: 'string', 'number', 'boolean'. Defaults to 'string'.
reversed	no	string	Apply mapping in reverse: 'true' or 'false'.

Example

Basic conversions

A conversion map “User” has been created with the following values:

- **case sensitive** : false
- **no match behavior** : "default"
- **default value** : "unknown_user"

name	alias
user1	new_user1
user2	new_user2

An event workflow is defined containing the convertField action with the following arguments:

Argument Name	Argument Value
mapName	User
field	custom_info.user
target	custom_info.updatedUser

Argument Name	Argument Value
type	string
reversed	false

Which the UI translates to:

```
{"mapName":"User","field":"custom_info.user", "target":
"custom_info.updatedUser", "type": "string", "reversed": "false"}
```

An event enters the workflow with the following `custom_info`:

```
{
  "user": "user1"
}
```

The action returns true and updates the event to have the `custom_info`:

```
{
  "user": "user1",
  "updatedUser": "new_user1"
}
```

If a new event is received with the `custom_info`:

```
{
  "user": "user3"
}
```

The action again returns **true** but updates the event using the conversion map default value:

```
{
  "user": "user1",
  "updatedUser": "unknown_user"
}
```

If the “User” conversion map had used the “exclude” no match behavior instead, the action would have returned **false** and the event would have remained unchanged.

If the “User” conversion map had used the “retain” no match behavior, the action would have returned **true** and the original value would have been copied into the ‘target’:

```
{
  "user": "user1",
  "updatedUser": "user1"
}
```

List conversions

If a field holds a list of primitive values, the conversion is applied to each element of the list.

Using the conversion map and workflow above, add an event with the following `custom_info`:

```
{
  "user": [ "user1", "user2", "user3" ]
}
```

Would result in the following results depending on the “no match” behavior:

retain

Returns **true** and updates `custom_info` to:

```
{
  "user": [ "user1", "user2", "user3" ],
  "updatedUser": [ "new_user1", "new_user2", "user3" ]
}
```

default

Returns **true** and updates `custom_info` to:

```
{
  "user": [ "user1", "user2", "user3" ],
  "updatedUser": [ "new_user1", "new_user2", "unknown_user" ]
}
```

exclude

Returns **true** and updates `custom_info` to:

```
{
  "user": [ "user1", "user2", "user3" ],
  "updatedUser": [ "new_user1", "new_user2" ]
}
```

In this last example, the returned list excludes the unconvertible value.

A special case for list conversion is when the field holds is a CSV list. The list is split into elements, which are converted and joined to produce a new CSV list, which may have fewer elements.

Casting target type

The conversion process treats all values as strings and produces string results. The "type" argument allows the results to be explicitly casted to either numbers or booleans. For core target fields, there is implicit casting.

For example, a conversion map “Severity” has been created with the following values:

- **case sensitive** : false
- **no match behavior** : "default"
- **default value** : "1"

name	alias
Critical	5
Major	4
Minor	3
Warning	2
Intermediate	1
Clear	0

An event workflow is defined containing the `convertField` action with the following arguments:

Argument Name	Argument Value
mapName	Severity
field	custom_info.severity
target	severity
type	
reversed	

Which the UI translates to:

```
{"mapName":"Severity","field":"custom_info.severity", "target": "severity"}
```

An event enters the workflow with the following `custom_info`:

```
{
  "severity": "warning"
}
```

The `convertField` action finds the string value “2” in the Severity conversion map. The target is “severity”, which is a core field that expects an integer value, so it is implicitly casted to a number and used to set the event severity.

The following core fields are implicitly casted:

Field	Event Types	Field Type
signature	event, alert, alertUpdate, alertClose	string
source_id	event, alert, alertUpdate, alertClose	string
external_id	event, alert, alertUpdate, alertClose	string
manager	event, alert, alertUpdate, alertClose	string
source	event, alert, alertUpdate, alertClose	string
class	event, alert, alertUpdate, alertClose	string
agent	event, alert, alertUpdate, alertClose	string
agent_location	event, alert, alertUpdate, alertClose	string
type	event, alert, alertUpdate, alertClose	string
description	event ,alert, alertUpdate, alertClose, sig, sigUpdate, sigClose	string
severity	event, alert, alertUpdate, alertClose	number
internal_priority	sig, sigUpdate, sigClose	number

Updating other core fields with the `convertField` action will fail.

For the custom_info, workFlowContext and payload fields, the type can be explicitly casted as required.

Casting to the number type

Fields containing single values will either result in a valid number equivalent to the string or the action will fail:

- “1” becomes 1 and the action returns **true**.
- “one” is just a string, won’t conversion and the action returns **false**.

For fields containing an array of values, any convertible strings become 0:

- [“1', “one”] becomes [1, 0] and the action will return **true**.

Casting to the boolean type

An empty string with any of the following becomes false: “false”, “0”, “-0”, “null”, “NaN”, “undefined”.

Any other non-empty string becomes true.

Casting CSV lists

For fields containing CSV lists, casting is applied after converting and joining to the new CSV string as a whole.

convertStringToList

A Workflow Engine function that converts a string of words separated by a defined character into a Javascript array (list).

You can use this function to overwrite the value of the source field with the resulting list or copy the resulting list into a given target field.

You can use any regular expression as the separator.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert and enrichment workflows.

This function is available for event, alert, and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `convertStringToList` takes the following arguments:

Name	Required	Type	Description
`sourceField`	Yes	String	Contains the original string.
`separator`	No	String	Separator character that divides the values in the string. Default separator character is comma.
`targetField`	No	String	Destination field for the resulting list. Default is the field entered for sourceField.

Example

The following example demonstrates typical use of Workflow Engine function `convertStringToList`.

You want to convert the string "`network :: database :: server :: operating system`" in `custom_info.services` to a list and you want to overwrite `custom_info.services` with the resulting list . Set the following:

- `sourceField`: custom_info.services
- `separator`: \s+::\s+
- `targetField`: custom_info.serviceList

The UI translates your settings to the following JSON:

```
{ “sourceField”:”custom_info.services” }
{ “separator”:“\s+::\s+” }
{ ”targetField”: custom_info.serviceList }
```

If the source field does not contain any value or the separator cannot be used, then this function will return `false`. Otherwise, the function will copy the following list `[ "network" , "database" , "server" , "operating system" ]` into the given target field.

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 [196] 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 [35].

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 'copyFromAlert-ToEvent' 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 [35].

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 [35].

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 [35].

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"}
```

copyFromEventToAlert

A Workflow Engine function that allows you to copy `custom_info` fields from an incoming event to the existing alert that the event would de-duplicate to. You can use the function to partially update custom info on an alert de-duplication. This can be used in the global custom info update preference configured within the AlertBuilder.

This action takes a list of fields to selectively copy between the event and alert. These fields have to be under `custom_info` .

The `custom_info` prefix is optional in the field list. For example, the following two fields lists would have the same functionality: copying `custom_info.location.city` from the event to `custom_info.location.city` in the associated alert:

```
[ "custom_info.location.city" ]
[ "location.city" ]
```

This action is useful when an incoming event's `custom_info` may change over the course of the alert's lifecycle and you need the updated values in subsequent functionality. For example to satisfy a workflow entry filter or to satisfy a tirgger or exclusion filter for a recipe.

If you want to merge the contents of `custom_info` upon every event de-duplication, then use the Alert Builder global `custom_info` merge. See the AlertBuilder Moobot comments, or contact your Cisco technical resource for more information.

The action returns true if all attempted updates are successful and false if one or more attempted updates fail. If the event has no associated alert the action returns false. For example, if the event will create a new alert.

NOTE
This action is a direct key replacement whereby values are replaced rather than merged. If the event is an array or object, it overwrites any existing alert value for the same key.

NOTE
This action results in an AlertUpdate issued for each field. Moolets that are `event_handler`-driven will receive each of these updates.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for event workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `copyFromEventToAlert` takes the following arguments:

Name	Required	Type	Description
fields	yes	object	The list of `custom_info` fields to selectively copy from the event to the alert. These do not need to be specified as `custom_info.field`, but can be.

Example

To selectively update an alert's custom info with the values from an incoming event for `custom_info.location.city`, `custom_info.supportGroup` and `custom_info.applications`:

- **fields** : ["location.city" , "applications"]

The UI translates your settings to the following JSON:

```
{"fields":["location.city","applications","custom_info.suppportGroup"]}
```

If the fields exist in the event, their values will be copied into the associated alert.

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 [35].

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 [35].

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:

1. You create two workflows: one to add data from `custom_info.location`, the other from `custom_info.datacentre`.
2. 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.
3. You use the `copyToPayload` function to copy from the correct key into the payload using the following actions:
   a. `createPayload [109]` configured with the map name (AlertExport).
   b. `copyToPayload` for alerts within the city, with `payloadKey` set to `"location"` and `value` set to `$custom_info.location.city`.
   c. `copyToPayload` with `payloadKey` set to `"currentTime"` and `value` set to `$moog_now`.
   d. `exportViaRest [129]` 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`.

createAlertOpsIncident

A Workflow Engine function that creates a new AlertOps alert for a situation.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `createAlertOpsIncident` has no arguments.

Example

This function, in conjunction with the AlertOps integration, sends a REST request to AlertOps to create a new alert from a Situation.

The integration adds a new workflow “Create AlertOps Incident” to the “Situation Integration” Workflow Engine containing this action.

createIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to create a new incident for the named service.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `createIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value alertops.
`instanceName`	no	string	An optional instance name for integrations supporting multiple instances. Default to the first instance.
`templateName`	no	string	A template name for integrations supporting predefined templates. If unspecified, a default template is used.
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

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 [35].

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 [129] or exportViaKafka [127].

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 [144] 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 [35].

Arguments

Workflow Engine function `createPayload` takes the following arguments:

Name	Required	Type	Description
`mapName`	Yes	String	Name of the map from the Payload Maps integration.

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 [35].

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 [35].

Arguments

Workflow Engine function `createTopology` takes the following arguments:

Name	Required	Type	Description
topology-Name	yes	string	The name or substited value for the topology. To substitute a value, use $(<attrib-ute_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:

- `topologyName`: my network
- `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 [35].

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:

- `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 [35].

Arguments

Workflow Engine function `deassignAlert` has no arguments.

dejaVu

A Workflow Engine function that allows you to determine if a workflow has processed a piece of data. You can optionally specify a time frame to limit the amount of time that the data was last seen. Deja vu is a French phrase that means "already seen."

This function returns `true` if the item has been seen previously within the time frame. It returns `false` if the item has not been processed or the lookup mechanism failed.

This function uses the Enrichment API to persist 'seen' values and allow 'seen' values to be seeded into the system. See Enrichment API.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert workflows.

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 [35].

Arguments

Workflow Engine function `dejaVu` takes the following arguments:

Name	Required	Type	Description
`key`	Yes	String	Source field to use as the key. Can be complex.
`value`	Yes	String	Source field containing the value to look up. Can be compound.
`age`	No	Number	Time to reset the 'last seen' value in seconds. Default is forever.
`enrich`	No	String	True or false. Specifies if the last seen data should be added to the object (under `custom_info.dejaVu.<key>`).

Example

The following example demonstrates typical use of Workflow Engine function `dejaVu`.

An application emits event messages containing an error code. You want to address these event so you add the alert to a Situation. You only want to add the first instance of the error message to a Situation and mark subsequent instances as 'error seen'. You also want to decrease the alert severity of subsequent instances.

Given the following event data:

```
"description" : "Application : APP001, Error code : 0001, a memory
allocation error has occurred."
```

In an event workflow you want to extract the “signature” of the error, the application identifier and the error code. Write the signature to the workflow context so you can use it later in the `dejavu` action. For example, create a `searchAndReplace` action with the following settings:

- **field**: description
- **expression**: Application : (.*), Error code : (\d+)
- **map**: {"workflowContext.errorCode":"$extract.2","workflowContext.appName":"$extract.1"}

The WorkflowEngine writes the following data to the workflow context:

```
{
    "errorCode": "0001",
```

```
    "appName": "APP001"
}
```

fNext, create a `dejavu` action to see if the error code has come up before. To qualify that the error codde is an an app error code within the same application, use a compound key comprised of the extracted data in the workflowContext. You can use the key mechanism to increase or decrease the scope/granularity of the dejavu action.

Set the following:

- **key**: AppErrorCodes::$(workflowContext.appName)
- **value**: $(workflowContext.errorCode)
- **age**: <leave blank>
- **enrich**: true

This action logic checks for the "errorCode 0001" for "application APP001." If true, it mark the event has having been seen. It updates the `custom_info` for the event will be enriched with the details: key, value, last seen as epoch and ISO date, and the dejavu flag.

If you want to limit the timeframe for the check, set the age to 86400 to limit the check to the past 24 hours.

The first time the action dectects the data, it writes the following `custom_info`. Note the `dejaVu` value is `false`:

```
       "dejavu": {
            "appErrorCodes::APP001": {
                "dejaVu": false,
                "lastSeen": 1591873839,
                "lastSeenDate": "2020-06-11T11:10:39.000Z",
                "value": "0001"
            }
        }
```

For subsequent events, the Workflow Engine sets the `deJaVu` to `true` and updates the value of `lastSeen`. For example:

```
    "dejavu": {
            "appErrorCodes::APP001": {
                "dejaVu": true,
                "lastSeen": 1591874042,
                "value": "0001",
                "lastSeenDate": "2020-06-11T11:14:02.000Z"
            }
        }
```

You can create subsequent actions to lower the severity and to change the description to indicate that the event has already been seen.

Because the previously seen data is stored in the Enrichment API database, you can use it to seed deny lists and and and allow lists. Instead of building complex filters to drop events, seed the correct key and value data into the enrichment database. Then you can create a single DejaVu based Workflow Engine for that “dictionary”. This allows an external system to control the allow/deny lists via the Enrichment API alone.

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 integration.

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 [35].

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:

- `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 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 [35].

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 [35].

Arguments

Workflow Engine function `deleteTopologyLink` takes the following arguments:

Name	Required	Type	Description
topology-Name	yes	string	The name or substited value for the topology. To substitute a value, use $(<attrib-ute_name>). For example $(custom_info.myTopology)
sourceNode	yes	string	The name or substituted value for the 'A' endpoint (source node). To substitute a val-ue, 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 [35].

Arguments

Workflow Engine function `deleteTopologyNode` takes the following arguments:

Name	Required	Type	Description
topology-Name	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 [267] 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 [35].

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 [35].

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 [35].

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 [266] 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 [35].

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 [35].

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 [35].

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 [109] 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 [109] 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 [35].

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 [109] 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 [109] 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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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.

getCorrelationInfo

A Workflow Engine function that checks for the existence of correlation info for a Situation. If true, the action sets a workflow context available for subsequent actions. If you don't specify the **contextName** argument, it uses the default context item `ciRelatedItems`.

If correlation info exists and the subscriber (serviceName) has interest then the context item, it sets an array of CI items and returns **true**. For example:

```
{
    [
      {
        "external_id":"e801043c2f319050c9d1dc1a2799bbbb",
        "sig_id":111,
        "service_name":"ServiceNow",
        "properties":"{\"service_name\":\"ServiceNow-WFE\",\"external_id
\":\"e801043c2f319050c9d1dc1a2799bbbb\"}"}]
      }
    ]
};
```

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `getCorrelationInfo` takes the following arguments:

Name	Required	Type	Description
serviceName	yes	string	External solution name. One of `ServiceNow`, `Remedy`, `Cherwell`, `JIRA`, `Assyst`, `PagerDuty`
contextName	no	string	Context name for retrieved CI. If not specified then defaults to `ciRelatedItems`.

Example

The following example demonstrates typical use of Workflow Engine function `getCorrelationInfo`. In this case we check for existence of correlation info for ticketing solution ServiceNow.

Set the following:

- **serviceName** : ServiceNow
- **contextName** :

The UI translates your settings to the following JSON:

```
{"serviceName":"ServiceNow"}
```

The action should be set to Forward the alert or situation as needed but in most cases it should be set to ‘Stop This Workflow’ on failure.

The function determines if correlation info exists for the specified target's Situation, returning false otherwise.

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 integration.

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 [35].

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 [35].

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 [35].

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"

See Enrich Alerts Using a JDBC Data Source 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:

- `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 [129] or exportViaKafka [127].

This function relates directly to the payload maps from your Payloads 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 [35].

Arguments

Workflow Engine function `getPayload` takes the following arguments:

Name	Required	Type	Description
`mapName`	Yes	String	Name of the map from the Payloads integration.

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 [35].

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 relates directly to the database and table definitions from your ServiceNow Enrichment integration.

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 [35].

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
  }
}
```

getThreadEntry

A Workflow Engine function that returns a thread entry for a thread in a Situation.

You can use this function to retrieve and store thread entries under the `threadEntry` key in the `workflowContext` field for use in a Situation workflow. You can only use this function with Situations that contain threads with an `entry_id` .

`getThreadEntry` returns true if a thread entry exists and has been successfully retrieved. Otherwise, this function returns false.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `getThreadEntry` has no arguments.

Example

The action is intended to be used in a SigAction Workflow Engine. You can use it after a sigActionFilter action to limit to specific sigAction codes, for example “Added Entry To Thread”. Used on its own it will retrieve the threadEntry for any sigAction that contains an entry_id.

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 [35].

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 [35].

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 [35].

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 [35].

Arguments

Workflow Engine function `hasNotMerged` has no arguments.

hasNotSweptUp

A Workflow Engine function that returns true if the workflow entry filter did not sweep up any alerts or Situations. If the trigger alert or Situation is in the associated (swept up) list, it is removed before the check is made.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `hasNotSweptUp` has no arguments.

hasPRCAlert

A Workflow Engine function that returns true if a Situation has at least one alert with a Probable Root Cause (PRC) value set (`rc_probability`).

Optionally, you can specify a minimum percentage value for the probability. If the top PRC alert from the situation has a probability less than this percentage, the action returns false.

If no alert has a PRC value or if PRC is not running, the action returns false.

This action uses the MoogDbv2 API call `getTopPrcDetails` with a limit of 1 to return the “top” alert. This limits the results to the alert with the highest probability.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `hasPRCAlert` takes the following arguments:

Name	Required	Type	Description
minPercent	no	number	The minimum probability % the returned alert needs for the action to return true. Validated as a number between 0-100

Example

You can use the `hasPRCAlert` function before a `createServiceTicket` function to prevent the Workflow Engine from creating a ticket for a Situation for Situations without a PRC alert. This is effective for cases where you require a PRC alert for ticketing or notification.

Optionally you can supply a minimum percentage for the PRC alert using the `minPercent` argument. The action can prevent subsequent functions like `createServiceTicket` from creating ticket until there is an alert with at least the specified percentage PRC. For example, you could set the **minPercent** to 50%:

- **minPercent** : 50

The UI translates your settings to the following JSON:

```
{"minPercent":50}
```

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 [35].

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 [35].

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.

hasSweptUp

A Workflow Engine function that returns true if the workflow entry filter swept up any alerts or Situations. If the trigger alert or Situation is in the associated (swept up) list, it is removed before the check occurs.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `hasSweptUp` has no arguments.

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

Arguments

Workflow Engine function `isOlderThan` takes the following arguments:

Name	Required	Type	Description
age	Yes	Number	Minimum object age in seconds.

isPrimaryTeamSet

A Workflow Engine function that returns true if the primary team for a Situation is set. You can optionally check the team name against the specified one. If the primary team has not been set, or does not match the specified name, the action returns false.

If the action returns true, adds the Team Name to `workflowContext.primaryTeam`.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `isPrimaryTeamSet` takes the following arguments:

Name	Required	Type	Description
`teamName`	no	string	The name of a team to check the primary team against.

Example

You can check if a Situation has a primary team set by not using the "teamName" parameter. If the primary team is set to any team name, the action returns true, returning false otherwise.

You can check if the primary team name matches a predefined team by using the teamName parameter. For example, to see if the primary team is "Networks:

- **teamName** : Networks

The UI translates your settings to the following JSON:

```
{"teamName":"Networks"}
```

If the primary team is set to “Networks", `isPrimaryTeamSet` returns true. If the primary team is not set or is not set to “Networks”, the action returns false.

If the action returns true, the primary team will be copied into the workflowContext under “primaryTeam” to allow the team name to be used in subsequent actions. For example, if a Situation Export has been set up, you could add the primaryTeam name to the payload.

Actions:

- `primaryTeamSet` : has the team been set?
- `getPayload` : create a predefined payload.
- `copyToPayload` : the key could be “primaryTeam” and the value would be a substitution from the workflowContext “$(workflowContext.primaryTeam)”.
- `exportViaRest` : export to a predefined endpoint using the modified payload to include the primary team.

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 [35].

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 [35].

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 [35].

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 [35].

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"] }
                }
```

listSituationAlertIds

A Workflow Engine function that adds the current alertIds in the Situation into the workflow context under “situationAlertIds”. By default, this is a JSON list (an array). Optionally, you can convert this information to a CSV string. You can use the resulting object in subsequent actions as needed. When used with a sweep up filter, the final list contains the unique alert ids from all in-scope Situations.

This action uses the underlying MoogDbV2 API call `getSituationAlertIds()`.

This function is available as a feature of the Add-ons v2.2 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 [35].

Arguments

Workflow Engine function `listSituationAlertIds` takes the following arguments:

Name	Required	Type	Description
`convertToCSV`	no	string (true|false)	Converts the resulting list to a CSV.

Example

The following example demonstrates typical use of Workflow Engine function `listSituationAlertIds`.

The resulting workflow context when using the default (list):

```
WORKFLOW CONTEXT: SITUATION: 186 :  :
{
    "situationAlertrIds": [
        569,
        570,
        571,
        532
    ]
}
```

When “convertToCSV” is set to true, the resulting workflow context is:

WORKFLOW CONTEXT: SITUATION: 186 : :

```
WORKFLOW CONTEXT: SITUATION: 186 :  :
{
    "situationAlertrIds": "569,570,571,532"
}
```

listSituationHosts

A Workflow Engine function that adds the current hostnames in the Situation into the workflow context under “situationHosts”. By default, this is a JSON list (an array). Optionally, you can convert this information to a CSV string. The resulting object or string can be used in subsequent actions as needed. When you use this function with a sweep up filter, the final list contains the unique hosts from all in-scope Situations.

The `listSituationHosts` function uses the underlying MoogDbV2 API call `getSituationHosts()`.

This function is available as a feature of the Add-ons v2.2 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 [35].

Arguments

Workflow Engine function `listSituationHosts` takes the following arguments:

Name	Required	Type	Description
`convertToCSV`	no	string (true|false)	Convert the resulting list to a CSV.

Example

The resulting workflow context when the default (list) is used:

```
WORKFLOW CONTEXT: SITUATION: 186 :  :
{
    "situationHosts": [
        "localhost",
        "10.0.1.2",
        "10.0.1.1"
    ]
}
```

When convertToCSV is set to true, the resulting workflow context is:

WORKFLOW CONTEXT: SITUATION: 186 : :

```
WORKFLOW CONTEXT: SITUATION: 186 :  :
{
    "situationHosts": "localhost,10.0.1.2,10.0.1.1"
}
```

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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":
["class","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 [35].

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"}
```

notifySlack

A Workflow Engine function that sends a request to Slack channel(s).

You can use this function to send a request only after you set the target Slack channel using the `setSlackTarget` function.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `notifySlack` has no arguments.

Example

To use the `notifySlack` function, set it as the last action in your workflow and assure that at a Slack channel has been defined in the `setSlackTarget` function.

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 [35].

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 [35].

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 [35].

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"}
```

processMicroFocusOOAutomationResponse

A Workflow Engine function that processes responses from Micro Focus OO Automation.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `processMicroFocusOOAutomationResponse` has no arguments.

Example

This function, in conjunction with the Micro Focus Operations Orchestration integration, handles responses back from Micro Focus OO containing the execution results from launched flows.

The integration adds a new workflow “Process Micro Focus OO Response” to both the “Situation Inform Engine” and “Alert Inform Engine” Workflow Engine containing this function. The function is “eventless” and can handle responses for both alerts and Situations.

Trigger the function using a request of this format:

```
curl -X POST -k -u ${grazeuser}:${grazepw} \
     ${aiopshost}/graze/v1/sendToWorkflow?refjectUnauthorized=false \
     -H "Content-Type: application/json" \
     -d \
     '{ "workflow_name":"Process Micro Focus OO Response",
        "engine_name":"Situation Inform Engine",
        "context":{
          "ceventType":"${ceventtype}",
          "ceventId":"${ceventid}",
          "instanceName":"${instance}",
          "status":"${status}",
          "summary":"${formattedResults}",
          "resultsLink":"${resultsLink}"
        }
      }'
```

The `${ceventtype}`, `${ceventid}` and `${instance}` variables are taken from the triggering request. The `${status}`, `${formattedResults}` and `${resultsLink}` variables update with the flow results.

receiveUpdateFromAlertOpsIncident

A Workflow Engine function that adds `recipients` to the corresponding AlertOps alert for teams added to a situation.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `receiveUpdateFromAlertOpsIncident` has no arguments.

Example

This function, in conjunction with the AlertOps integration, handles notifications back from AlertOps.

The integration adds a new workflow “Handle AlertOps Response” to the “Situation Integration” Workflow Engine containing this action.

removeItemsFromList

A Workflow Engine function that removes a set of specified items from an existing list (array) and writes the resulting modified list back to the source field, or to a different destination field. The source and destination fields can be located in the workflowContext or an event, alert or Situation field. The “itemsToRemove” can include an explicit list, or a substituted value. If the substituted value contains a list already, this will be expanded (refer to the Examples, below, for more information).

This function is available as a feature of the Add-ons v2.2 download and later.

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 [35].

Arguments

Workflow Engine function `removeItemsFromList` takes the following arguments:

Name	Required	Type	Description
`sourceField`	yes	string	The source field (CEvent or workflowContext) for the list.
`itemsToRemove`	yes	object	The list of items to remove from the source list. Substitution is allowed.
`destinationField`	no	string	The field to write the results back to. Can be either the CEvent field or the workflowContext.

Example

The following example demonstrates typical use of Workflow Engine function `removeItemsFromList`.

The following examples use this source list custom_info.services:

```
[ "apple" , "pear" , "skippy" , "boomerang" ]
```

Removing “in place” from a CEvent field:

- `sourceField` : custom_info.services
- `itemsToRemove` : [“apple”]

The UI translates your settings to the following JSON:

```
{"sourceField":"custom_info.services","itemsToRemove":["apple"]}
```

Removing from one list and writing to another:

- `sourceField` : custom_info.services
- `itemsToRemove` : [“apple”]
- `destinationField` : custom_info.newServices

The UI translates these settings to the following JSON:

```
{"sourceField":"custom_info.services","itemsToRemove":
["apple"],"destinationField":"custom_info.newServices"}
```

Using substitution in the “itemsToRemove”

Given a sourceField of workflowContext.source and a destination field of workflowContext.newService, we will remove a dynamic list also held in the workflowContext.

The `workflowContext` before the action:

```
WORKFLOW CONTEXT: ALERT: 634 :  :
{
    "source": [
        "apple",
        "pear",
        "dog",
        "cat",
        "banana"
    ],
    "delete": [
        "banana"
    ]
}
```

In this example, to delete “banana” from the source.

The `itemsToRemove` argument uses a substitution, but this still has to be an object, such as [ ] (a list, or array). The workflow performs the substitution as needed and ensures that the list is correctly formed, using the standard substitution syntax.

For example:

```
{"sourceField":"workflowContext.source","itemsToRemove":["$
(workflowContext.delete)"],"destinationField":"workflowContext.newServices"}
```

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 [35].

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 [35].

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".

resolveIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to resolve an incident for the named service.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `resolveIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration, one of: none
`instanceName`	no	string	An instance name for integrations supporting multiple instances. Defaults to the first instance
`templateName`	no	string	A template name for integrations supporting predefined templates. If unspecified, a default template is used.
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

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 [108] 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 [35].

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 [35].

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 [97] 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 [35].

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 [35].

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 [35].

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":"NOC"
}}
```

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 [198] 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 [35].

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.

sendAcknowledgedToIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to acknowledge an incident for the named service.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendAcknowledgedToIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration, one of: none
`instanceName`	no	string	An instance name for integrations supporting multiple instances. Default to the first instance.
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

sendAlertsToWorkflow

A Workflow Engine function that sends Situation alerts to a named workflow within a named Inform based workflow engine using the MoogdbV2 `sendToWorkflow` API call. Returns true when all the alerts are sent successfully to the target workflow. Returns false if one or more alerts fails to send.

You can specify unique alerts. See the `getSituationAlertIds` MoogDb V2 API call for details: getSituationAlertIds.

You can optionally provide a context for the target workflow. This context is available to the target workflow in the workflowContext.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendAlertsToWorkflow` takes the following arguments:

Name	Required	Type	Description
`engineName`	yes	string	The name of the target inform based engine.
`workflowName`	yes	string	The name of the workflow within the target inform engine.
`uniqueOnly`	no	string (true|false)	This argument should be used if you want to send only alerts unique to this Situation. Otherwise sends all alerts.
`context`	no	object	A JSON object ({…}) containing the context you want passed into the target workflow's workflowContext. By default the triggering situation id is passed within the context.

Example

When a Situation first has a severity of "Critical", send all the contained alerts to the "Alert Inform Engine" and execute a workflow named "Export Alerts". Pass the situation description and severity to the target workflow. The context parameter can use the substitution macro syntax to populate details about the triggering situation.

- **engineName** : "Alert Inform Engine"
- **workflowName** : "Export Alerts"
- **uniqueOnly** : false
- **context** : { "description" : "$(description)" , "severity" : "$EXPAND(internal_priority)" }

The UI translates your settings to the following JSON:

```
{"engineName":"Alert Inform Engine","workflowName":"Export
Alerts","uniqueOnly":"false","context":
{"severity":"$EXPAND(internal_priority)","description":"$(description)"}}
```

sendAssignedToAlertOpsIncident

A Workflow Engine function that assigns the corresponding AlertOps alert to a user corresponding to the situation moderator.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendAssignedToAlertOpsIncident` has no arguments.

Example

This function, conjunction with the AlertOps integration, forwards moderator assignment to AlertOps. If a user mapping is added for the moderator in the integration tile and the “Assign on moderator assignment” option is selected, then the corresponding AlertOps alert is assigned to the mapped user. Otherwise, a message is added as a "reply" to the AlertOps alert.

The integration adds a new workflow “Assign AlertOps Incident” to the “Situation Integration” Workflow Engine containing this function.

sendCIsAddedToIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to add a list of CIs to the incident for the named service. By default, the action expects a list of CIs in the newCIs workflowContext key. You can use the optional CIs argument to override the default.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendCIsAddedToIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value alertops.
`instanceName`	no	string	An instance name for integrations supporting multiple instances. Defaults to the first instance
`CIs`	no	string	A list of CIs. Example: "node1", "node2"
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

Example

Within a Situation Delta workflow, this action can be proceeded by a `haveSituationCIsChanged` function. The `haveSituationCIsChanged` function returns true when the CIs in a situation have changed, and it populates the newCIs workflowContext key with a list of CIs added to the situation.

If the forwarding behavior on the `haveSituationCIsChanged` function is set to “Stop This Workflow”, the subsequent `sendCIsAddedToIncident` function only runs when the CIs in the situation have changed. It then takes the list of new CIs added from the workflowContext, so the CIs argument can be omitted.

sendEmail

A Workflow Engine function that sends an email message to a list of recipients using an email server instance defined in the Email Endpoints integration. You can specify the recipients, subject, and message body for the email in the function arguments.

The `sendTo` argument accepts the following:

- A comma separated list of email addresses.
- The contents of an alert field. For example: `$(custom_info.eventDetails.emailTo)`.
- For Situations only: a team mapping from the Email Endpoints integration using the `$EXPAND(teams)` macro. This requires you to configure email for the team mapping in the Email Endpoints integration. See sendEmailUsingTemplate [212] for an example.

Use `$NL` or `<br>` to specify line breaks in the message body.

The Add-ons package includes some example workflows to help you get started.

This action is for email only.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendToEmail` takes the following arguments:

Name	Required	Type	Description
`emailConfig`	yes	string	The **Server Config Name** from the Email Endpoints integration. It contains the connection information for your email server.
`sendTo`	yes	string	The email destination. A comma separated list of email addresses, the contents of an alert field, or a team maping.
`subject`	yes	string	Email subject. Supports macro substitution.
`message`	yes	string	Email message body. Supports macro substitution.

Example

The following example demonstrates typical use of Workflow Engine function `sendToEmail`. The configuration refers to an Email Endpoints integration configuration called "Example."

Set the following:

- **emailConfig** : Example
- **sendTo** : tester@example.com, example@example.com
- **subject** : AIOps ACTION REQUIRED :: $EXPAND(internal_priority) situation has been raised - please review
- **message** : A $EXPAND(internal_priority) situation has been raised $NL $NL Description - $(description) $NL $NL Service(s) affected are $EXPAND(services) $NL $NL AIOps situation link :: $CONTEXT_URL(config[servlets.conf][webhost])

The UI translates your settings to the following JSON:

```
{"emailConfig":"Example","sendTo":"tester@example.com,example@example.com","
subject":"AIOps ACTION REQUIRED :: $EXPAND(internal_priority) situation has
been raised - please review","message":"A $EXPAND(internal_priority)
situation has been raised $NL $NL Description - $(description) $NL $NL
```

```
Service(s) affected are $EXPAND(services) $NL $NL AIOps situation link ::
$CONTEXT_URL(config[servlets.conf][webhost])"}
```

The `sendToEmail` action forwards alerts or Situations as needed. It is set to "Stop This Workflow" on failure. Returns false if the request payload is not successfully built.

sendAssignedToIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to assign the incident for the named service. By default, the function takes the user name from the Situation moderator or alert owner but you can use the moderatorName argument to assign a different user name.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendAssignedToIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value alertops.
`instanceName`	no	string	An instance name for integrations supporting multiple instances; defaults to the first instance.
`moderatorName`	no	string	The moderator name to use.
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

Example

Within a Situation Delta workflow, this action can be preceded by a sigActionFilter action with the actionTypes set to '["Assigned Moderator"]'. The function returns true if the workflow receives an "Assigned Moderator" Situation action.

If the forwarding behavior of the sigActionFilter function is set to “Stop This Workflow”, the subsequent `sendAssignedToIncident` function only runs when the workflow receives an "Assigned Moderator" Situation action.

To forward the moderator assignment to AlertOps, give the following arguments to the `sendAssignedToIncident` function:

Argument Name	Argument Value
`serviceName`	alertops
`instanceName`	
`moderatorName`	
`arguments`	

Which the UI translates to:

```
{"serviceName":"alertops" }
```

This sends a Situation Inform to the “Assign AlertOps Incident” workflow in the Situation Integration WFE. This workflow contains a `sendAssignedToAlertOpsIncident` function which uses the moderator_id of the in-scope situation to retrieve the name of the new moderator and forward a message to AlertOps.

Alternatively, if the `sendAssignedToIncident` action is given the following arguments:

Argument Name	Argument Value
`serviceName`	alertops
`instanceName`	
`moderatorName`	$EXPAND(moderator_id)

Argument Name	Argument Value
arguments	

Which the UI translates to:

```
{"serviceName":"alertops", "moderatorName":"$EXPAND(moderator_id)" }
```

The function sends a Situation Inform to the “Assign AlertOps Incident” workflow in the Situation Integration WFE. This workflow contains a `sendAssignedToAlertOpsIncident` function which uses the moderator workflowContext key to retrieve the name of the new moderator and forward a message to AlertOps.

sendEmailUsingTemplate

A Workflow Engine function that sends an email message to a list of recipients using an email server instance defined in the Email Endpoints integration. You can specify the recipients of the email, but the email subject and message body are defined in a template in the Email Endpoints integration.

The `sendTo` argument accepts the following:

- A comma separated list of email addresses.
- The contents of an alert field. For example: `$(custom_info.eventDetails.emailTo)`.
- For Situations only: a team mapping from the Email Endpoints integration using the $EXPAND(teams) macro. This requires you to configure email for the team mapping in the Email Endpoints integration.

Use `$NL` or `<br>` to specify line breaks in the message body.

The Add-ons package includes some example workflows to help you get started.

This action is for email only.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendToEmailUsingTemplate` takes the following arguments:

Name	Required	Type	Description
`emailConfig`	yes	string	**Server Config Name** from the Email Endpoints integration. It contains the connection information for your email server.
`messageTemplate`	yes	string	**Message Template Name** from the Email Endpoints integration. THe template contains the message subject and body rules for the email message.
`sendTo`	yes	string	The email destination. A comma separated list of email addresses, the contents of an alert field, or a team maping.

Example

The following example demonstrates typical use of Workflow Engine function `sendEmailUsingTemplate`. The configuration refers to an Email Endpoints integration configuration called "Example." THe email destination depends on a team mapping in the Email Endpoints integration. For example the team ‘Cloud Devops’ is mapped to "devops@example.com".

Set the following:

- **emailConfig** : testing
- **messageTemplate** : defaultSituation
- **sendTo** : $EXPAND(teams)

The defaultSituation template defines 2 settings for the email to be sent, e.g:

- **Message Subject** : AIOps Notification :: A $EXPAND(internal_priority) situation has been raised
- **Message Body** : A $EXPAND(internal_priority) situation has been raised
 \n Description - $(description) \n \n Service(s) affected are $EXPAND(services) \n \n AIOps situation link :: $CONTEXT_URL(config[servlets.conf][webhost])

The UI translates your settings to the following JSON:

```
{"emailConfig":"Example","messageTemplate":"defaultSituation","sendTo":"$EXP
AND(teams)"}
```

The action should be set to forward the alert or situation as needed but in most cases it should be set to ‘Stop This Workflow’ on failure.

The function returns false when the request payload was not built successfully.

sendMicroFocusOOAutomationRequest

A Workflow Engine function that sends a request to Micro Focus Operations Orchestration to launch a flow.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendMicroFocusOOAutomationRequest` has no arguments.

Example

This function, in conjunction with the Micro Focus Operations Orchestration integration, and sends a REST request to a configured Micro Focus Operations Orchestration instance to launch a flow.

You can then trigger these workflows using the `sendRequestToAutomation` function from an alert of situation workflow in order to launch Micro Focus Operations Orchestration flows.

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 [35].

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.

sendNoteToIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to post a message to the incident for the named service. By default, the function uses the contents of workflowContext noteText key. You can override this behavior using the either of the optional `noteText` or `templateName` arguments.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendNoteToIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value alertops.
`instance-Name`	no	string	An instance name for integrations supporting multiple instances; defaults to the first instance.
`template-Name`	no	string	An integration template which populates the messages. The function uses the `noteText` key when there is no `templateName` or `noteText` argument.
`noteText`	no	string	Text which posts when `noteText` is included and `templateName` is omitted.
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

sendRequestToAutomation

A Workflow Engine function sends a request to the corresponding outbound automation workflow to invoke specified action for the named service.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendRequestToAutomation` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the automation, one of: puppet, ansible, bonitasoft
`instanceName`	no	string	An instance name for integrations supporting multiple instances. Defaults to the first instance
`templateName`	no	string	A template name for integrations supporting predefined templates. If unspecified, a default template is used.
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

sendSituationsToWorkflow

AA Workflow Engine function that sends the active Situation an alert is a member of to a named workflow within a named Inform based workflow engine using the MoogdbV2 `sendToWorkflow` API call. Returns true if all the Situations were sent successfully to the target workflow. Returns false if one or more Situations failed to send.

You can optionally provide a context for the target workflow. This context is available to the target workflow in the `workflowContext`. By default, adds the triggering alert_id to the context as `workflowContext.alert_id`.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendSituationsToWorkflow` takes the following arguments:

Name	Required	Type	Description
engineName	Yes	String	The name of the target inform based engine.
workflowName	Yes	String	The name of the workflow within the target inform engine.
context	No	Object	A JSON object ({…}) containing the context to pass to the target workflow's `workflowContext`. By default , passes the triggering alert id within the context.

Example

The following example demonstrates typical use of Workflow Engine function `sendSituationsToWorkflow`.

Set the following:

- `engineName`: “Situation Inform Engine”
- `workflowName`: “Add Thread”
- `context`: { “description” : “$(description)” , “severity” : “$EXPAND(severity)”, “count” : $TO_INT(count) }

The UI translates your settings to the following JSON:

```
{"engineName":"Situation Inform Engine","workflowName":"Add
Thread","context":{"severity":"$EXPAND(severity)","description":"$
(description)". “count” : $TO_INT(count) }}
```

sendTeamsAddedToAlertOpsIncident

A Workflow Engine function that adds recipients to the corresponding AlertOps alert for teams added to a situation.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendTeamsAddedToAlertOpsIncident` has no arguments.

Example

This action, in conjunction with the AlertOps integration, adds situation team additions to AlertOps. If a team mapping is added for the teams in the integration tile and the “Assign on team additions” option is selected, then the mapped AlertOps Recipient Groups are added to the corresponding AlertOps alert. Otherwise, a message is added as a "reply" to the AlertOps alert.

The integration will add a new workflow “Add Teams to AlertOps Incident” to the “Situation Integration” Workflow Engine containing this action.

sendTeamsAddedToIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to add a list of teams to the incident for the named service. By default, the teams to add are taken from the teamsAdded workflowContext key. You can override this by using the optional `teamNames` argument.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendTeamsAddedToIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value alertops.
`instanceName`	no	string	An instance name for integrations supporting multiple instances. Default to the first instance.
`teamNames`	no	string	A list of team names. Example: teamA, teamB
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

Example

Within a Situation Delta workflow, this action can be preceded by the `getTeamsAdded` function. This function returns true if teams were added to a situation and have populated the teamsAdded workflowContext key with a list of teams.

If the forwarding behavior of the `getTeamsAdded` function is set to “Stop This Workflow”, the subsequent `sendTeamsAddedToIncident` function only runs when the workflow receives a Situation containing added teams.

To forward the team assignment to AlertOps, we can give the following arguments to the `sendTeamsAddedToIncident` function:

Argument Name	Argument Value
`serviceName`	alertops
`instanceName`	
`teamNames`	
`arguments`	

Which the UI translates to:

```
{"serviceName":"alertops" }
```

This sends a Situation Inform message to the “Add Teams to AlertOps Incident” workflow in the Situation Integration WFE. This workflow contains a `sendTeamsAddedToAlertOpsIncident` function which uses the teamsAdded workflowContext key to retrieve the names of the new teams and forward a message to AlertOps.

Alternatively, if the sendTeamsAddedToAlertOpsIncident action were given the following arguments:

Argument Name	Argument Value
`serviceName`	alertops
`instanceName`	
`teamNames`	TeamA, TeamB

Argument Name	Argument Value
arguments	

Which the UI translates to:

```
{"serviceName":"alertops", "teamNames":"TeamA, TeamB" }
```

In this second example, the function sends a Situation Inform to the “Add Teams to AlertOps Incident” workflow in the Situation Integration WFE, but now the teamsAdded context key contains the list: [“TeamA”, “TeamB”], so these team names are used to send the update to AlertOps.

sendTeamsRemovedToIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to remove a list of teams from the incident for the named service. By default, the teams to remove are obtained from the teamsRemoved workflowContext key. Use the optional `teamNames` argument to override the default behavior.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendTeamsRemovedToIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value alertops.
`instanceName`	no	string	An instance name for integrations supporting multiple instances. Defaults to the first instance
`teamNames`	no	string	A list of team names. Example: TeamA, Team B
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

sendThreadEntryToAlertOpsIncident

A Workflow Engine function that sends a reply to the corresponding AlertOps alert.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendThreadEntryToAlertOpsIncident` has no arguments.

Example

This function, in conjunction with the AlertOps integration, forwards collaboration thread entries to AlertOps as replies to the corresponding AlertOps alert.

The integration adds a new workflow “Post to AlertOps Incident” to the “Situation Integration” Workflow Engine containing this action.

sendThreadEntryToIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to update the incident for the named service using thread entry retrieved by a preceding `getThreadEntry` function.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `sendThreadEntryToIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value alertops.
`instanceName`	no	string	An instance name for integrations supporting multiple instances; defaults to the first instance.
`prependText`	no	string	Text to prepend to the thread entry message.
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

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 [233] 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 [35].

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 [233] 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 [233] 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 for more information.

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 [234] 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 [35].

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 [234] 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 [244] 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 [35].

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 [244] 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 [35].

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 [109] 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 [109] 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 [35].

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 Reference 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 [35].

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 [35].

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 [35].

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 yourAnsible 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 [225] 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 [35].

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 [226] 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 [35].

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 [35].

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 [238] or setCustomInfoJSONValue [237] functions. For example set the `agent_location` field to "London".

This function is available for event workflows only.

Back to Workflow Engine Functions Reference [35].

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 [238] 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 [35].

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 [237] 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 [35].

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 [35].

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 integration.

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 [35].

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 integration.

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 [35].

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 [35].

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 [35].

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 yourPuppet 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 integration.

`setPuppetAutomation` typically precedes a sendToPuppet [227] 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 [35].

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 [35].

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 [35].

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"
    ]
}
```

setSituationServices

A Workflow Engine function that explicitly sets the Situation Impacted Services. The list of services to set can be explicit or can use the standard argument substitution. Optionally, you can replace the existing services with the specified list. By default, the specified services are added to the existing Impacted Services.

Services added to the system by this process have the description “Automatically added service”.

This function is available as a feature of the Add-ons v2.2 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 [35].

Arguments

Workflow Engine function `setSituationServices` takes the following arguments:

Name	Required	Type	Description
`services`	yes	object	A list of services to add. Can include substitutions. If the substitution is a list then the action will expand this. Example: `[ $(custom_info.services) ]`

Example

The following example demonstrates typical use of Workflow Engine function `setSituationServices`.

To set the Impacted Services to include (add) “myService”:

- `services` : [“myService”]
- `replaceServices` : false

The UI translates your settings to the following JSON:

```
{"services":["myService"],"replaceServices":"false"}
```

To add a substituted value (in this case the value of `custom_info.service`) in the highest PRC alert:

- `services` : [$PRC($(custom_info.service))]
- `replaceServices` : false

The UI translates this to:

```
{"services":["$PRC($(class))"],"replaceServices":"false"}
```

To replace the current services with a list of services from an existing list:

Given a `custom_info` field “service”:

```
{
  "services" : "mail","network"]
}
```

- `services` : [“$(custom_info.services)”]
- `replaceServices` : true

The function expands this list into the the underlying list of services and applies these to the Situation.

The UI translates this to:

```
{"services":[ "$(custom_info.services")],"replaceServices":"true"}
```

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 [35].

Arguments

Workflow Engine function `setSituationState` takes the following arguments:

Name	Required	Type	Description
state	Yes	String	State to set, for example TICKETED.

setSlackTarget

A Workflow Engine function that sets the target Slack channel and title for a message payload. Before you use this function, you must set up the **Sack Incoming Webhook** in the Slack integration.

You can use this function to build that message payload to send to Slack channels.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `setSlackTarget` has no arguments.

Workflow Engine function `setSlackTarget` takes the following arguments:

Name	Required	Type	Description
`slackChannel`	Yes	String	The target Slack channel.
`title`	No	String	Message that to send to the specified Slack channel. Can be static text and/or Macros.

Example

The following example demonstrates typical use of Workflow Engine function `setSlackTarget`.

You want to message payload will be built using integration configuration for the Slack channel called aiopstesting and override the default title with your own. Set the following:

- `channel`: aiopstesting
- `title`: $EXPAND(internal_priority) level Situation - $(description)

The UI translates your settings to the following JSON:

```
{"slackChannel":"aiopstesting","title":"$EXPAND(internal_priority) level
Situation - $(description)"}
```

If you do not define a title, then this function will use the default that is set under the target channel's integration configuration for the alert or Situation.

The function forwards the alert or Situation once the message payload is successfuly build. Otherwise, the function returns `false` if it cannot successfully build the message payload.

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 [35].

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 [35].

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 [35].

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 [35].

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 [35].

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 [260] 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 [35].

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 [35].

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 [35].

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 [35].

Arguments

Workflow Engine function `staticLookup` takes the following arguments:

Name	Required	Type	Description
key	Yes	String	Source field to use as the key.
lookup-Name	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 [35].

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.

testSendEmail

A Workflow Engine function that specifies an email server instance as defined in the integrations UI. The integration contains the email addresses, email subject, and email message to use for the email request..

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `testSendEmail` takes the following arguments:

Name	Required	Type	Description
`emailConfig`	yes	string	TBD Same as in UI

Example

The following example demonstrates typical use of Workflow Engine function `testSendEmail`.

Set the following:

•

The UI translates your settings to the following JSON:

updateAlertOpsIncident

A Workflow Engine function that sends generic updates to the AlertOps alert.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for Situation workflows only.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `updateAlertOpsIncident` has no arguments.

Example

This function, in conjunction with the AlertOps integration, sends generic updates to AlertOps.

The integration adds a new workflow “Update AlertOps Incident” to the “Situation Integration” Workflow Engine containing this action.

updateIncident

A Workflow Engine function that sends a request to the corresponding outbound integration workflow to perform a generic update to an incident for the named service.

This function is available as a feature of the Add-ons v2.2 download and later.

This function is available for alert and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `updateIncident` takes the following arguments:

Name	Required	Type	Description
`serviceName`	yes	string	The name of the incident integration; has the value of `alertops`.
`instanceName`	no	string	An instance name for integrations supporting multiple instances. Default to the first instance.
`templateName`	no	string	A template name for integrations supporting predefined templates. If unspecified, a default template is used.
`arguments`	no	object	A JavaScript object that passes any additional arguments as context to the workflow and adds them to the existing workflow context.

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 [35].

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 [35].

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 [35].

Arguments

Workflow Engine function `willDeduplicateAlert` has no arguments.

workflowContextSearchAndReplace

A Workflow Engine function that allows you to use regular expressions to extract values from the workflow context and add them to an `workflowContext.extract` object. The data is available for use in subsequent actions.

This function is available as a feature of the Add-ons v2.1 download and later. See Install Cisco Add-ons [12] for information on how to upgrade.

This function is available for event, alert, and Situation workflows.

Back to Workflow Engine Functions Reference [35].

Arguments

Workflow Engine function `workflowContextSearchAndReplace` takes the following arguments:

Name	Required	Type	Description
`field`	yes	string	The name of the workflowContext field to apply the expression (extraction) to.
`expression`	yes	string	A regular expression with subgroups to apply to the field.

Example

Given a starting workflow context:

```
{
    "application": "app001",
    "linkEstablished": "Resource server001 connected to resource db001",
}
```

Set the following arguments in the `workflowContextSearchAndReplace` action:

- **field** : application
- **expression** : Resource (.*) connected to resource (.*)

The workflow context would contain the following after the action ran:

```
{
    "application": "app001",
    "linkEstablished": "Resource server001 connected to resource db001",
    "extract": {
        "1": "server001",
        "2": "db001"
    }
}
```

`workflowContext.extract` would contain the extracted subgroup values from the expression. These values can be used in subsequent actions through substitution (e.g. $(worfklowContext.extract.1) would be substituted by “server001”).

AlertOps Integration

This is a reference for the AlertOps UI integration.

This integration enables bidirectional communication with an external AlertOps instance:

- Creates new AlertOps alerts from Situations
- Passes Situation updates to AlertsOps:
  - Situation collaboration thread entries added as replies to the AlertOps alert
  - Teams added to a Situation can be used to assign new Recipient Groups in AlertOps
  - Situation moderator can be used to assign the AlertOps alert to a corresponding AlertOps user
  - Updates can be forwarded to AlertOps on Situation changes
  - Resolving the Situation closes the corresponding AlertOps alert
- Handles notifications back from AlertOps
  - Receives a callback with the AlertOps alert id when a new alert is created
  - Closes the Situation when the AlertOps alert closes
  - Adds a Situation collaboration thread entry when a reply is added to an AlertOps alert
  - Acknowledges the Situation and adds a collaboration thread entry when the AlertOps alert is assigned to a user

Before you begin

Before you start to set up your integration, ensure you have met the following requirements:

- An AlertOps integration **APIKey**
- The AlertOps notify and api base URLs, if different from the defaults
- The Source and Source Name to use with the integration
- A new graze API user to use with the integration to enable acknowledgment of Situations when the alert is assigned in AlertOps
- Details of AlertOps users and Recipient groups you wish to map from Cisco Crosswork Situation Manager users and teams

The Source and Source Name should match values used in the AlertOps Inbound integration. The notify and api base URLs and the APIKey can be obtained from the AlertOps instance.

Configure the AlertOps Integration in Cisco Crosswork Situation Manager

NOTE
Refer to https://help.alertops.com/en/ for additional configuration information.

To configure the AlertOps integration in Cisco Crosswork Situation Manager:

1. Navigate to the **Integrations** tab.
2. In the Ticketing section, click **AlertOps**.
3. Provide a unique **integration** name. You can use the default name or customize the name according to your needs.
4. Provide a unique **Instance Name**. This name is used at the Source Name in AlertOps. You can add additional instances using the **Add Instance** button.
5. Add the AlertOps **APIKey** and update the **Integration User Name** and **Source** as required.
6. Make sure that the **Base Notify URL** and **Base API URL** in the “AlertOps Endpoints” setting matches correct values for the AlertOps instance. The default values should work for most SaaS instances.
7. Confirm to save the setting and enable the integration.

The integration deploys a number of Integration Workflow Engine workflow to enable bidirectional communication with AlertOps. These are disabled initially.

You must also configure the AlertOps instance to handle the interaction with Cisco Crosswork Situation Manager. Refer to the following section for detailed information.

See AlertOps Workflow [278] for details of the workflow and usage of the integration.

Configure AlertOps

To enable the integration, you must also configure AlertOpt to handle the incoming request from Cisco Crosswork Situation Manager and to trigger notifications back. These details are fully documented by AlertOps but are also captured here for reference.

1. Create new Custom Alert Fields

From **Administration** > **Custom Alert Fields**, click **Add Alert Type** to create a new Alert Type:

Alert Type: Cisco Alert

Group	Alert Fields	Data Type
Cisco	ai_aiops_instance	ShortString
Cisco	ai_description	LongString
Cisco	ai_last_event_time	Numeric
Cisco	ai_last_state_change	Numeric
Cisco	ai_severity	ShortString
Cisco	ai_sitn_id	Numeric
Cisco	ai_sitn_url	LongString
Cisco	ai_status	ShortString
Cisco	ai_total_alerts	Numeric

These are the basic fields required by the integration and can be extended but should be removed.

2. Create an Escalation Rule

If required, create a new Escalation Rule by navigating to **Escalation Rules** and then clicking **Add Escalation Rule**. If a suitable escalation rule exists, use it in the following steps.

The Escalation Rule must use the new Alert Type, created above. This is updated on the “Options” tab of the Escalation Rule.

3. Create a new Inbound Integration

Navigate to **Integrations** > **Inbound Integrations**, click **Add API Integration** and then click the **Custom** tile to open the dialog for a new custom integration. On the “Basic Settings” tab, add an Integration Name, select the Escalation Rule and check the **Enabled** checkbox.

URL Mapping

On the “Advanced Settings” tab, click **URL Mapping** and add the following:

Option	Value	Comments
Short Text	<description>	If additional fields are added to Situations, they can be used here.
Subject Text	<description>	If additional fields are added to Situations, they can be used here.
Source Name	aiops_instance	Populated with the AlterOps integration instance name. Uncheck the the **Static** checkbox alongside the “Source Name” option.
Source	Cisco	Must match the “Source” entered into the Cisco Crosswork Situation Manager AlertOps Integration tile.
Long Text	<description>	If additional fields are added to Situations, they can be used here.
Method	POST	
Source ID	sitn_id	
Source URL	sitn_url	
Source Status	status	
Open Alert When	Contains Any [“Opened”]	
Close Alert When	Contains Any [“Closed”, “Resolved”]	

Option	Value	Comments
Update Alert When		

Advanced Mapping

Next, click **Advanced Mapping**. Populate the Custom Alert Fields from the inbound integration by entering the following:

Option	Value
ai_sitn_id	sitn_id
ai_last_event_time	last_event_time
ai_severity	severity
ai_total_alerts	total_alerts
ai_last_state_change	last_state_change
ai_aiops_instance	aiops_instance
ai_description	description
ai_status	status
ai_sitn_url	sitn_url

A default Recipient Group can be set here but the Dynamic Recipient Group section can be used for more fine grained control of recipients.

Dynamic Recipient Groups

On the “Dynamic Recipient Groups” you can map incoming Cisco Crosswork Situation Manager team names to AlertOps Recipient Group.

These teams are those added to a Situation when the Situation is forwarded to AlertOps and are populated into a list in the “teams” field of the AlertOps inbound integration. You can add conditions to teams assigned to AlertOps Recipient groups based on the names of the Cisco Crosswork Situation Manager teams listed in the “teams” field.

4. Add methods to the Outbound Integrations

Navigate to **Integrations** > **Outbound Integrations**, click **Add Outbound Integration** to open the “Outbound Integration Detail” dialog, and then enter the following:

Option	Value	Comments
Password		The password for the Cisco Crosswork Situation Manager Graze user
UserName	alertops	A suitable Cisco Crosswork Situation Manager Graze user
Integration Name	Cisco sendToWorkflow	
Web Security Type	Basic	

Click **Save and Continue** to add methods to the outbound integration.

Methods

Add the following methods:

Table 1. alertCreated

Option	Value	Comments
URI	https://<server base url>//graze/v1/sendToWorkflow	Use the base URL for your Cisco Crosswork Situation Manager UI instances.
Name	alertCreated	
Type	REST	
Web Method	POST	

Option	Value	Comments
Alert Type	Cisco Alert	Matches the Alert Type from step 1.
Request Type	JSON	
Response Data Type	JSON	

Request Data:

```
{
   "engine_name": "Situation Integration",
   "workflow_name": "Handle AlertOps Response",
   "sitn_id": "<<MessageThread.SourceIdentifier>>",
   "context": {
      "alertops_id": "<<MessageThread.MessageThreadID>>",
      "subject": "alertCreated",
      "passedInstance": "<<Attribute.ai_aiops_instance>>"
   }
}
```

Table 2. messageAdded

Option	Value	Comments
Name	messageAdded	
Type	REST	
Alert Type	Cisco	Should match the Alert Type from step 1.
URI	https://<server base url>//graze/v1/sendTo-Workflow	Use the base URL for your Cisco Crosswork Situation Manager UI instances.
Request Type	JSON	
Web Method	POST	
Response Data Type	JSON	

```
{
   "engine_name": "Situation Integration",
   "workflow_name": "Handle AlertOps Response",
   "sitn_id": "<<MessageThread.SourceIdentifier>>",
   "context": {
      "subject": "messageAdded",
      "message": "<<Message.MessageText>>",
      "passedInstance": "<<Attribute.ai_aiops_instance>>"
   }
}
```

Table 3. sendAssigned

Option	Value	Comments
Name	sendAssigned	
Type	REST	
Alert Type	Cisco Alert	Matches the Alert Type from step 1.
URI	https://<server base url>//graze/v1/sendTo-Workflow	Use the base URL for your Cisco Crosswork Situation Manager UI instances.
Request Type	JSON	
Web Method	POST	
Response Data Type	JSON	

Request Data:

```
{
   "engine_name": "Situation Integration",
```

```
   "workflow_name": "Handle AlertOps Response",
   "sitn_id": "<<MessageThread.SourceIdentifier>>",
   "context": {
      "subject": "alertAssigned",
      "user": "<<MessageThread.OwnerUserName>>",
      "passedInstance": "<<Attribute.ai_aiops_instance>>"
   }
}
```

Table 4. alertClosed

Option	Value	Comments
Alert Type	Cisco Alert	Matches the Alert Type from step 1.
Name	alertClosed	
Request Type	JSON	
Response Data Type	JSON	
Type	REST	
URI	http://<server base url>//graze/v1/sendTo-Workflow	Use the base URL for your Cisco Crosswork Situation Manager UI instances.
Web Method	POST	

Request Data:

```
{
   "engine_name": "Situation Integration",
   "workflow_name": "Handle AlertOps Response",
   "sitn_id": "<<MessageThread.SourceIdentifier>>",
   "context": {
      "alertops_id": "<<MessageThread.MessageThreadID>>",
      "subject": "alertClosed",
      "passedInstance": "<<Attribute.ai_aiops_instance>>"
   }
}
```

5. Add workflows

New workflows are required to trigger outbound communication back to Cisco Crosswork Situation Manager. These are added by navigating to “Workflows” and clicking **Add Workflow**.

Cisco Initial Response

The workflow triggering the initial callback to Cisco Crosswork Situation Manager when an alert is opened. This workflow used to update the originating Situation with the AlertOps Alert ID.

Options	Value	Comments
Name	Cisco Initial Response	
Type	Message Thread	
Alert Type	Cisco Alert	Matches the Alert Type from step 1.
Enabled	Checked	
Scheduled	Unchecked	

Table 5. Match All Conditions

Attribute	Name	Condition
Standard	MessageThreadStatusType	is Open

Table 6. Match Any Condition

Attribute	Name	Condition
NA	NA	NA

Table 7. Actions

Action	Value	Comment
Outbound Service Notfication	Cisco sendToWorkflow alertCreated	NA

Cisco Send Assigned

The workflow that notifies Cisco Crosswork Situation Manager when an AlertOps alert is assigned. This workflow adds a thread entry to the Situation and may acknowledge and assign to the Cisco Crosswork Situation Manager AlertOps integration user.

Options	Value	Comments
Name	Cisco Send Assigned	
Type	MessageThread	
Alert Type	Cisco Alert	Matches the Alert Type from step 1.
Enabled	Checked	
Scheduled	Unchecked	

Table 8. Match All Conditions

Attribute	Name	Condition
Standard	MessageThreadStatusType	is Assigned

Table 9. Match Any Condition

Attribute	Name	Condition
NA	NA	NA

Table 10. Actions

Action	Value	Comment
Outbound Service Notification	Cisco sendToWorkflow - sendAssigned	NA

Cisco Send Closed

The workflow that notifies Cisco Crosswork Situation Manager when an AlertOps alert is assigned. This workflow resolves the originating Situation.

Options	Value	Comments
Name	Cisco Send Closed	
Type	MessageThread	
Alert Type	Cisco Alert	Matches the Alert Type from step 1.
Enabled	Checked	
Scheduled	Unchecked	

Table 11. Match All Conditions

Options	Value	Comments
Standard	MessageThreadStatusType	is Closed

Table 12. Match Any Condition

Attribute	Name	Condition
NA	NA	NA

Table 13. Actions

Action	Value	Comment
Outbound Service Notification	Cisco sendToWorkflow - alertClosed	NA

Cisco Send Comment

The workflow that sends a notification to Cisco Crosswork Situation Manager when a reply to an AlertOps alert is made. This workflow adds a thread entry to the originating Situation in Cisco Crosswork Situation Manager.

Options	Value	Comments
Name	Cisco Send Comment	
Type	Message	
Alert Type	Cisco Alert	Matches the Alert Type from step 1.
Enabled	Checked	
Scheduled	Checked	
Recurrence Interval	0	

Table 14. Match All Conditions

Name	Condition	Comment

Table 15. Match Any Condition

Attribute	Name	Condition
NA	NA	NA

Table 16. Actions

Action	Value	Comment
Outbound Service Notification	Cisco sendToWorkflow - messageAdded	NA

6. Add workflows to Escalation Rule

The final step to enable the integration in AlertOps is to assign the outbound workflow to the escalation rule use by the inbound integration.

Navigate to **Escalation Rules** and then select the rule in use by the inbound Integration configured in step 3. On the “Workflows” tab, click **Add Workflow** and move each of the following from the list of available workflow to the list of selected workflow:

- Cisco Initial Response
- Cisco Send Assigned
- Cisco Send Closed
- Cisco Send Comment

Click **Update** to commit the changes and then click **Update** again in the main Escalation Rule dialog.

AlertOps Workflows

This is a reference for the workflows integrated with the AlertOps UI integration.

Workflows

The AlertOps integration installs the following workflows:

Outbound Notifications

Workflows which perform outbound notification.

Workflow Name	Engine Name	Description
Create AlertOps Incident [278]	Situation Integration	Sends an outbound notification to AlertOps to create a new alert.
Close AlertOps Incident [279]	Situation Integration	Sends an outbound notification to AlertOps to close an alert.
Update AlertOps Incident [279]	Situation Integration	Sends an outbound notification to AlertOps to update an alert.
Post to AlertOps Incident [280]	Situation Integration	Sends an outbound notification to AlertOps to add a reply to an alert.
Assign AlertOps Incident [280]	Situation Integration	Sends an outbound notification to AlertOps to assign an alert.
Add Teams to AlertOps Incident [281]	Situation Integration	Sends an outbound notification to AlertOps to add a Recipient Group to an alert.

Inbound Notifications

Workflows which handle inbound notifications.

Workflow Name	Engine Name	Description
Handle AlertOps Response [281]	Situation Integration	Updates situations with responses from AlertOps.

Create AlertOps Incident

Usage: Automated creation of AlertOps Alerts

To enable automated creation of new AlertOps alert when new Situations are created, confirm the “Create AlertOps Incident” outbound notification workflow is enabled.

To trigger the outbound notification, add a workflow to the “Situation Workflows” WFE that includes the “createIncident” action. For example:

Workflow Name: Create AlertOps Incident

Entry filter: 'status' != 'Closed'

First Match Only: Checked

Action Name	Function	Arguments	Forwarding Behavior
120 seconds	Delay	120 seconds	
Trigger sends to AlertOps	createIncident	`{"serviceName":"alertops", "instanceName":"AlertOps1"}`	Stop This Workflow

Where the instanceName argument matches the Instance Name configured in the AlertOps integration tile.

This sends a request to AlertOps and add a thread entry:

```
Sending request to open AlertOps alert: <succeeded|failed>
```

You can adjust the Entry filter in the trigger workflow to control which Situations are forwarded to AlertOps.

The outbound notification uses the AlertOps Integration payload map in the AlertOps integration to send data from the Situation to AlertOps. Add new fields to this map to pass additional custom attributes.

If the payload map is updated, changes are required in AlertOps to the Custom Alert Fields and inbound integration.

Close AlertOps Incident

Usage: Automated closing of AlertOps Alerts

To enable the automated closing of AlertOps alerts when a Situation is resolved, confirm the “Close AlertOps Incident” outbound notification workflow is enabled.

To trigger the outbound notification, add a workflow to the “Situation Workflows” WFE that includes the “closeIncident” action. For example:

Workflow Name: Close AlertOps incidents

Entry filter: 'status' is one of '["Closed","Resolved"]'

First Match Only: Checked

Action Name	Function	Arguments	Forwarding Behavior
0 seconds	Delay	0 seconds	
Trigger send to AlertOps	closeIncident	`{"serviceName":"alertops", "instance-Name":"AlertOps1"}`	Stop This Workflow

Where the instanceName argument should match the Instance Name configured in the AlertOps integration tile.

This sends a request to AlertOps and adds a thread entry:

```
Sending request to close AlertOps alert: <succeeded|failed>
```

You can adjust the Entry filter in the trigger workflow to control which situations are forwarded to AlertOps.

The outbound notification uses the AlertOps Integration payload map in the AlertOps integration.

Update AlertOps Incident

Usage: Forwarding updates to AlertOps

To enable forwarding of Situation updates to AlertOps when the CIs in a situation change, confirm the “Update AlertOps Incident” outbound notification workflow is enabled.

To trigger the outbound notification a workflow should be added to the “Situation Delta” WFE that includes the “updateIncident” action.

Workflow Name: Notify CI Change to AlertOps Incident

The “Update AlertOps Incident” outbound notification uses the AlertOps update API to send a generic update to an AlertOps alert. Other trigger conditions can be added to the Situation Delta WFE to call this workflow if required.

The outbound notification uses the AlertOps Update payload map in the AlertOps integration to send data from the situation to AlertOps. Add new fields to this map to pass additional custom attributes.

If the payload map is updated, changes are required in AlertOps to the Custom Alert Fields and inbound integration.

Post to AlertOps Incident

Usage: Forwarding collaboration thread entries to AlertOps

To enable forwarding of situation collaboration thread entries to AlertOps, confirm the “Post to AlertOps Incident” outbound notification workflow is enabled.

To trigger the outbound notification, add a workflow to the “Situation Delta” WFE that includes the “sendThreadEntryToIncident” action. For example:

Workflow Name: Post to AlertOps Incident

Action Name	Function	Arguments	Forwarding Behavior
0 seconds	Delay	0 seconds	
Check action	sigActionFilter	`{"actionTypes":["Added Entry To Thread"]}`	
Get thread entry	getThreadEntry	-	
Trigger send to AlertOps	sendThreadEntryToIncident	`{"serviceName":"alertops", "instanceName":"AlertOps1", "prependText":"Collaboration post: "}`	

Where the instanceName argument matches the Instance Name configured in the AlertOps integration tile. The prependText argument is an optional prefix that can be added to the replies. In this case, it is the string “Collaboration post: “.

New collaboration thread entries will appears as replies to the AlertOps alert in the format:

```
Collaboration post: <Cisco Crosswork Situation Manager user name>:: <thread entry text>
```

Assign AlertOps Incident

Usage: Automated assignment of AlertOps alerts

To enable notification of situation assignment to AlertOps, confirm the “Assign AlertOps Incident” outbound notification workflows is enabled.

To trigger the outbound notification, add a workflow to the “Situation Delta” WFE that includes the “sendAssignedToIncident” action. For example:

Workflow Name: Assign AlertOps Incident

Action Name	Function	Arguments	Forwarding Behavior
0 seconds	Delay	0 seconds	-
Situation assigned	sigActionFilter	`{"actionTypes":["Assigned Moderator"]}`	Stop This Workflow
Trigger send to AlertOps	sendAssignedToIncident	`{"serviceName":"alertops", "instanceName":"AlertOps2", "moderatorName":"$EXPAND(moderator_id)"}`	Stop This Workflow

Where the instanceName argument matches the Instance Name configured in the AlertOps integration tile. The moderatorName argument is optional. If omitted, the moderator name is taken from the Situation retrieved by the outbound workflow.

By default, this workflow adds a reply to the AlertOps alert in the format:

```
Moogsoft situation assigned to moderator <moogsoft username>
```

The workflows can also automatically assign the AlertOps alert to an AlertOps user if:

- The “Assign on Moderator Assignment” option is checked in the AlertOps integration.
- The “User” Conversion Map in the AlertOps integration is enabled and updated with mappings from Cisco Crosswork Situation Manager usernames to AlertOps usernames

If a valid User mapping doesn’t exist for the new situation moderator, the workflow falls back to sending a reply message.

The default behavior in the absence of a match is “exclude”, which means the conversion fails and defaults to sending a reply message instead. Change this default behavior only when the usernames in both systems are identical or if a suitable default user account exists.

Add Teams to AlertOps Incident

Usage: Automated addition of recipient to AlertOps alerts

To enable notification when teams are added to a situation, confirm the “Add Teams to AlertOps Incident” outbound notification workflow is enabled.

To trigger the outbound notification, add a workflow to the “Situation Delta” WFE that includes the “sendTeamsAddedToIncident” action. For example:

Workflow Name: Add Teams to AlertOps Incident

Action Name	Function	Arguments	Forwarding Behavior
0 seconds	Delay	0 seconds	-
Get Teams added	getTeamsAdded	-	Stop This Workflow
Trigger sends to AlertOps	sendTeamsAddedToIncident	`{"serviceName":"alertops", "instanceName":"AlertOps1"}`	Stop This Workflow

Where the instanceName argument matches the Instance Name configured in the AlertOps integration tile.

By default, this workflow adds a reply to the AlertOps alert in the format:

```
Teams added to Moogsoft situation: <csv list of moogsoft team names>
```

The workflows can also automatically assign the AlertOps alert to an AlertOps user if:

- The “Assign on Moderator Assignment” option is checked in the AlertOps integration.
- The “User” Conversion Map in the AlertOps integration is enabled and was updated with mappings from Cisco Crosswork Situation Manager usernames to AlertOps usernames

If a valid User mapping doesn’t exist for the new situation moderator, the workflow falls back to sending a reply message.

In the absence of a match, the default behavior is “exclude”, which means the conversion fails and defaults to sending a reply message instead. Change this default behavior only when the usernames in both systems are identical or if a suitable default user account exists.

Handle AlertOps Response

Usage: Update Situation with AlertOps alert ID

To allow responses from AlertOps to update situations, confirm the “Handle AlertOps Response” workflow is enabled.

AlertOps uses an outbound integration to send the Alert ID back to Cisco Crosswork Situation Manager in response to a request to create a new alert.

This results in a new thread entry:

```
Created AlertOps alert <Alert ID>
```

An update to custom_info.ticketing:

```
{
  "ticketNumber": <Alert ID>
  "ticketStatus": "open"
}
```

Usage: Resolve Situation when AlertOps Alert Closes

AlertOps uses an outbound integration to notify Cisco Crosswork Situation Manager when the AlertOps alert is closed.

This results in a new thread entry:

```
AlertOps alert <Alert ID> is closed
```

An update to custom_info.ticketing:

```
{
  "ticketNumber": <Alert ID>
  "ticketStatus": "closed"
}
```

And the situation is moved to a resolved state.

Usage: Add thread entry for a reply to an AlertOps alert

AlertOps uses an outbound integration to notify Cisco Crosswork Situation Manager when a reply is added to an AlertOps alert.

This results in a new thread entry:

```
AlertOps message: <alertops reply message>
```

Usage: Acknowledge Situation when AlertOps Alert is Assigned

AlertOps uses an outbound integration to notify Cisco Crosswork Situation Manager when the AlertOps alert is assigned.

This results in a new thread entry:

```
Alert in AlertOps assigned to user: <alertops username>
```

Additionally, if a valid “Integration Username” is supplied in the integration and the situation isn’t already assigned, it is automatically acknowledged and assigned to the Integration user.

Micro Focus Operations Orchestration

The Micro Focus Operations Orchestration (also called Micro Focus HP Operations Orchestration) integration allows Cisco Crosswork Situation Manager to initiate flows in MicroFocus Operations Orchestration and for MicroFocus Operations Orchestration to pass back the flow status on completion.

A flow can be initiated:

- Manually, using a tool from a Situation or alert.
- Automatically, from a Situation or alert workflow.

Communication of flow status is passed back from MicroFocus Operations Orchestration by adding a callback into the flows.

Before you begin

Before you start to set up your integration, ensure you have met the following requirements:

- The REST endpoint URL for your Micro Focus Operations Orchestration instance.
- A username and password that allows flows to run.
- Details of flows you want to run (flowUuid, runName, logLevel, inputs).

Configure the Micro Focus Operations Orchestration Integration

To configure the Micro Focus Operations Orchestration integration:

1. Navigate to the **Integrations** tab.
2. Click **Micro Focus Operations Orchestration** in the Automation section.
3. Provide a unique integration name. You can use the default name or customize the name according to your needs.
4. Provide a unique **Instance Name**. You can add additional instances using the **Add Instance** button.
5. Update the Instance URL username and password for the Micro Focus Operations Orchestration instance.
6. Leave “Wait for Automation Request Response” set to “No”. Changing it forces requests to be performed synchronously, which slows down processing.
7. Customize the “Automation Status Failure Strings” to match failure messages that are returned by callback.
8. Select whether to add collaboration posts for Situation level automations.
9. Update the proxy settings, if required, to reach the Micro Focus Operations Orchestration instance.
10. Update the Flow Payload Templates sections with details of the flows you want to run.
11. Confirm to save the settings and enable the integration.

The integration deploys a number of Inform Workflow Engine workflows to enable bidirectional communication with Micro Focus Operations Orchestration.

The flows in the Micro Focus Operations Orchestration instance require updating to add a callback to send execution results back to Cisco Crosswork Situation Manager.

Once ready, see the Micro Focus Operations Orchestration Workflow for details of the workflow and usage of the integration.

Micro Focus Operations Orchestration Workflows

The Micro Focus Operations Orchestration integration installs the following workflows:

Outbound Requests

Workflows which perform outbound notification

Workflow Name	Engine Name	Description
Launch Micro Focus OO Flow	Alert Inform Engine	Sends an outbound request to Micro Focus OO to launch a flow
Launch Micro Focus OO Flow	Situation Inform Engine	Sends an outbound request to Micro Focus OO to launch a flow

Inbound Requests

Workflows which handle inbound notifications

Workflow Name	Engine Name	Description
Process Micro Focus OO Response	Situation Inform Engine	Processes an inbound request from Micro Focus OO to update a triggering alert or situation with execution status of the launched flow.
Process Micro Focus OO Response	Alert Inform Engine	Processes an inbound request from Micro Focus OO to update a triggering alert or situation with execution status of the launched flow.

Usage

The following sections explain how to use the Micro Focus OO Flows.

Automated Launching of Micro Focus OO Flows

To automatically launch flows from alerts or situations, ensure that the “Launch Micro Focus OO Flow” workflows are enabled.

To trigger the outbound request from an alert or situation, add a workflow to the “Alert Workflows” or “Situation Workflows” WFE that includes the `sendRequestToAutomation` function.

For example, the following added to the “Alert Workflows” WFE triggers the a flow to perform a ping test for alerts matching:

Workflow Name: Send Request to MFOO automation

Entry filter: 'manager' = 'MICROFOCUS'

First Match Only: Checked

Action Name	Function	Arguments	Forwarding Behavior
0 seconds	Delay	0 seconds	
Trigger request to MFOO	`sendRequestToAutomation`	`{"serviceName":"microfocusoo", "instanceName":"MicroFocusOO1", "templateName":"runPingTest"}`	Stop This Workflow

Where the `instanceName` argument matches the Instance Name configured in the AlertOps integration tile and the `templateName` matches a template configured in that instance.

If successful, the the alert updates `custom_info.automation.MicfoFocusOO.MicroFocusOO1` to:

```
{
  "status":"pending"
}
```

If the request is unsuccessful, the “status” and any returned error message will be added to the “summary”:

```
{
  "status":"failed",
  "summary": "Status code: 404, status_msg: Not Found, msg:  ......"
}
```

Manually Launching Micro Focus OO Flows

A client tool “Launch Micro Focus OO Flow” is available which allows the manual launching of flows from alerts and situations.

To use the tool, ensure that the “Launch Micro Focus OO Flow” workflows are enabled and update the tool prompts to include any instances and flow templates configured in the integration tile.

Processing Responses from Micro Focus OO

To update alerts and situations with the results of a flow executed in Micro Focus OO, ensure the “Process Micro Focus OO Response” is enabled and that the flows are updated to include the following callback:

```
curl -X POST -k -u ${grazeuser}:${grazepw} \
     ${aiopshost}/graze/v1/sendToWorkflow?refjectUnauthorized=false \
     -H "Content-Type: application/json" \
     -d \
     '{ "workflow_name":"Process Micro Focus OO Response",
        "engine_name":"Situation Inform Engine",
        "context":{
          "ceventType":"${ceventtype}",
          "ceventId":"${ceventid}",
          "instanceName":"${instance}",
          "status":"${status}",
          "summary":"${formattedResults}",
          "resultsLink":"${resultsLink}"
        }
      }'
```

The `${ceventtype}`, `${ceventid}` and `${instance}` variables are taken from the triggering request. The `${status}`, `${formattedResults}` and `${resultsLink}` variables update with the flow results.

The returned results are used to update `custom_info.automation.MicfoFocusOO.${instance}` of the originating alert or situation.

For an alert that belongs to no active situations, `custom_info.automation.MicfoFocusOO.${instance}` updates with:

```
{
  "status":"${status}",
  "summary": "${formattedResults}",
  "resultsURL": "${resultsLink}"
}
```

It the alert belongs to active situations, the summary is omitted from custom_info and instead adds a collaboration thread entry to the situations, if that option is enabled in the integration. In this case, `custom_info.automation.MicfoFocusOO.${instance}.alert-${ceventid}` in the situations also updates with:

```
{
  "status":"${status}"
}
```

For situations, `custom_info.automation.MicfoFocusOO.${instance}` will be updates with:

```
{
  "status":"${status}",
  "resultsURL": "${resultsLink}"
}
```

And a collaboration thread entry is added to the summary , if that option is enabled in the integration.