Event Correlation

opEvents does not just automatically suppress duplicate events (stateful or custom-matched); it can also create new events based on correlating recent event occurrences.
In versions 2.0.4 and newer you can use fine-grained controls to deal with the triggering events, and from version 2.2 onwards the contents of synthetic events are configurable, too.

This page describes how to configure event correlation.

General Configuration

This event correlation and synthesis feature is configured in the same way as the duplicate suppression, namely by putting event creation rules into `conf/EventRules.nmis`.

An event synthesis rule consists of:

- an event `name`, which specifies the name of the newly created event,
- a list of `events` (more precisely, their names), which are the events to consider for correlation,
- a (minimum) `count` of events that have to be detected to trigger the rule,
- an optional list of `groupby` clauses, which define whether the count is interpreted globally for all named events, or separately within smaller groups,
- optional `delayedaction` and `autoacknowledge` clauses, which define how the triggering events should be handled,
- an optional `enrich` clause, which adjusts the content of the newly created event,
- from version 2.2 onwards, optional `copy_first`, `copy_last`, `copy_highest` and `copy_groupby` clauses which further control the contents of the newly created event,
- from version 2.2 onwards, an optional `inhibit` parameter, which disables correlation temporarily after a rule has fired,
- and finally a `window` parameter, which defines the time window to examine.

(If you compare suppression and synthesis rules closely, you'll see that the main difference is the lack of a `suppress` clause for synthesis rules, whereas the suppression rules don't have `enrich` or `copy_*` clauses.)

Here is an example rule:

```
'3' => {
        name => 'Customer Outage',
         events => ["Node Down","SNMP Down"],
         window => '60',
         count=> 5,
         groupby=>['node.customer'], # count separately for every observed value of customer
         enrich=>{priority => 3, answer => 42}, # any such items gets inserted in the new event
},
```

This rule causes opEvents to look for Node Down and SNMP Down events in the last 60 seconds, separate them into per-customer groups (see grouping below); if it counts 5 or more such events in a group, then a new event called Customer Outage is created.

Grouping

If no `groupby` clause is present, then potential trigger events are selected solely by event name and event time (within the window), without any further scope limiting, i.e. globally across all nodes. For many common scenarios this may be too broad a selection; for example creating new events for a particular customer or service group only wouldn't be possible.

Grouping solves this problem: the set of potential triggering events is split into groups with matching property values and the `count` threshold is applied to those groups.

The `groupby` clause has the form of a list of `node.X` or `event.Y` property specifications (e.g. `node.customer` or `node.group`), which are used to group events into buckets for counting: only events that share the same values for all the listed grouping properties will be counted together.

For example, the `groupby` clause `[ 'node.customer', 'event.priority' ]` would cause this correlation rule to be applied independently for all combinations of customer and event priority. The clause given in the example block above will create a Customer Outage event *for any individual customer* with 5 outages in 60 seconds; without the `groupby` any 5 outages anywhere would cause a synthetic event to be created.

The `groupby` clause can make use of all common node properties which are listed here and the standard event properties which are documented on this page. Please note, however, that only event properties that were set during the event parsing stage are accessible when correlation is performed. For example policy actions can change an event (e.g. tagging, script execution) but policy actions are performed *after* correlation.

Event Content and Enrichment

Before opEvents version 2.2, synthetic events are always cloned from the most recent triggering event, then they get a new name from the synthesis rule name, and finally any static `enrich` clauses are evaluated. Synthetic events could **not** be stateful events, i.e. they were not subject to deduplication and could not be acknowledged (or 'closed') by any later 'opposite' event.

In version 2.2 this limitation has been removed, and much more precise control of the event content is possible.

Content Control Directives (Version 2.2 and newer)

When a synthesis rule creates a new event, the following steps are performed:

1. If no `copy_first`, `copy_last`, `copy_highest`, `copy_present` or `copy_groupby` directives are present, then a backwards-compatible directive '`copy_last => [ qr// ]`' is added.
2. (opEvents 2.2 and newer only) `copy_present` is evaluated first. It specifies which properties should be set *from their first occurrence*.
   This rule must contain explicit property names only, i.e. no regular expressions.
   opEvents checks all trigger events in chronological order, and when it finds an event that has a value for the desired property, it copies that value over and stops looking for that property. Any later events that might have the property as well do not contribute to the result.
   A rule like `copy_present => [ 'alpha', 'beta' ]` will pull the `alpha` and `beta` properties from wherever they are present for the first time, but independent of each other: a trigger event can contribute none, either or both properties.
3. `copy_first` is evaluated next, and specifies which event properties should be copied over from the *earliest trigger event.*
   Each listed property is copied over; if the directive contains a regular expression (e.g. `qr/cust.*/`, then all properties with names matching the regular expression are copied.
4. `copy_last` is checked next, and properties listed here are copied over from the *most recent trigger event*.
   The property copying does overwrite all properties that were set earlier (by `copy_first`).
5. `copy_highest` is checked next, and its properties are sourced from the *trigger event with the highest priority*.
   Again overwriting of properties may happen.
6. `copy_groupby` controls whether any of the grouping property values should be saved in the new event.
   The format is different for this directive: It must be a list of property target names (or the word 'undef'), in the same order as the `groupby` directive. For each element in the `groupby` list, the value of the grouping property is saved as the target name in the new event, if a target name is available in the `copy_groupby` list.
   If no `groupby` is given for this rule, then a `copy_groupby` directive has no effect.
7. Now the `enrich` clause is checked, and each of its property name - value pairs indicates which properties should be set to (or overwritten with) a particular static value.
8. Now the `node`, `stateful` and `element` properties are automatically adjusted if required (see below for details).
9. Finally, the event name is set to the rule name, certain undesirable properties are removed, an audit trail of triggering events is added (by adding the properties `nodes` and `eventids`), the event is marked as `synthetic` and is inserted into the database.

Please note that "earliest event" in step 2, 3 and 4 refers to the event with the earliest event timestamp, which does *not necessarily* reflect its processing order. opEvents processes inputs mostly - but not always - in chronological order. If you have multiple 'earliest' events (all with the same timestamp) then their order is undefined and `copy_first` will pick a random event. The same caveat applies for the "most recent event".

Automatic Event Node for Synthetic Events

If no `copy_*` or `enrich` clause has caused the the `node` property to be *set explicitly*, then the global default node is used instead.

'Set explicitly' means a `copy_*` or `enrich` clause did include the `node` property, i.e. **not** if the node property copying happened because of a regular expression.

The global default node in opEvents 2.2 is configurable using the configuration item `opevents_correlation_node`, and it's normally called "global". This virtual node is automatically (re)created if missing.

This behaviour is different from opEvents before 2.2, where all synthetic events were attached to the last trigger event's node. To emulate the old behaviour you have to change your correlation rules, so that they include the directive

```
copy_last => [qr//, 'node']
```

which causes a blanket copy of all properties from the last trigger event and an explicit copy of the node property (to disable the automatic event node choice).

Example Rule

Here is an example rule demonstrating the new directives:

```
'1' => {
   name => "Very Sick Node",
   events => [ "Node Down", "SNMP Down", "Interface Down", "Service Down",
               "Service Degraded", "Interface Flap", "Node Flap", "WMI Down" ],
   window => 120,
   count => 3,
   groupby => [ 'node.name' ], # we want separate events for each node of course
   enrich => { stateful => "Very Sick Node", priority => 5, state => 'down', element => undef }, # new event is
stateful only if stateful is set or copied by name
   copy_last => [ qr//, 'node' ], # can set from node here (all events share it)
   copy_groupby => [ 'node' ], # or from here; must set it explicitely somewhere, or the event goes to
opevents_correlation_node
},
```

Stateful Synthetic Events (Version 2.2 and newer)

By default, synthetic events are **not** stateful events, i.e. they are not subject to deduplication and they cannot be acknowledged (or 'closed') by any future 'opposite' event.

However, in 2.2 and newer it is possible to enable stateful handling for synthetic events:

1. Your rule must *explicitly* set the `stateful` property.
   Copying with a regular expression in `copy_*` does not meet this requirement, and a thusly copied `stateful` property is deleted before event creation.
2. Your rule must ensure that a suitable `state` property value is present.
3. Your rule should ensure that a suitable `element` property value is present, or opEvents will automatically create one from `groupby` information if that is available.
   As described in the documentation for Stateful Events, the combination of `node`, `stateful` and `element` properties must uniquely identify the stateful 'thing', and the value of the `state` property describes the new state.

The example rule above shows how a stateful 'very sick node' event can be created: the node name is set from the grouping criteria (i.e. all related triggers share the same node name), the `stateful` property is set with a static `enrich` clause, and there is no `element`, so at most one 'very sick node' stateful thing can exist for a single node.

If we wanted to acknowledge this event from a different correlation rule, we'd have to ensure that `node`, `stateful` and `element` properties with the same value are generated, but the `state` would have to be 'up' or 'closed' or 'ok'.

Here is another example, for a group-level stateful event:

```
8 => {
    name=>'sick group',
    events=>["Service Down","SNMP Down", "Node Down"],
    groupby => [ 'node.group' ],
    window=>150,
    count=>3,
    enrich => { stateful => "sick group", state => "down" },  # node will be opevents_correlation_node, element
will be group
    copy_last => [ qr// ],
},
9 => {
    name=>'happy group',
    events=>["Service Up", "some nice event" ],
    groupby => [ 'node.group' ],
    window=>300,
    count=>1,
    enrich => { stateful => "sick group", state => "up" },
    copy_last => [ qr// ],
},
```

Rule 8 specifies that three of the listed 'down' events in a single group should cause a new event that sets the 'sick group' state to down for this one group; the `element` property is auto-generated from the `groupby` data, and all such events are attached to the virtual node 'global'. Any repeat 'sick group' events would be statefully deduplicated. Because element is set to the group in question, every single group would have its own 'sick group' state.

As soon as a single positive event from the list in rule 9 arrives, the 'sick group' event is acknowledged and closed.

Event Processing for Synthetic Events

At this point the new event is inserted into the database, and is ready for further action processing. This action processing (e.g. escalation, mail notification, custom logging) is performed immediately.

Handling of the Triggering Events

Please note that this feature is only available in opEvents 2.0.4 and newer.

- If your rule contains a `delayedaction` clause (with a numeric value), then all *potentially triggering* events will have their action processing delayed by the given number of seconds.
  This affects **all events** whose name is in the event list of your rule, no matter whether the limits for triggering a synthetic event have been met or not. The `delayaction` value should therefore be set to a relatively small value.
- If your rule has the property `autoacknowledge` set to `"true"` or 1, then all triggering events will be automatically acknowledged and all action processing for them will be aborted.

The combination of these two controlling properties provides fine-grained storm control and the ability to create "combination events" that subsume and close any number of triggering events:

"Plain" Synthetic Events

If your rule sets neither `delayedaction` nor `autoacknowledge`, then the incoming potential trigger events will be processed as per normal and immediately, and any policy actions for them will be taken as soon as possible (but possible after being delayed by the `state_flap_window` - see Deduplication and storm control in opEvents for details). The trigger events are thus completed, and visible as current/unacknowledged, completely independent of any synthetic events that might get triggered by them later.

"Combination" Events

If your rule sets `delayedaction` (and optionally `autoacknowledge`), then the incoming events are delayed and held for the given time before any policy actions are taken for them. (The `delayedaction` setting should be the same as or larger than the rule's `window` setting.)

If the requirements for a synthetic event are met during that time, then the new synthetic event can "combine" and supersede the triggering events. If `autoacknowledge` is set, then all the triggering events will be acknowledged, closed and no actions will be taken for them at all. (Without `autoacknowledge` the triggering events would not have actions performed but would remain unacknowledged.)

The net effect is that the current events view would show only the new synthetic event as 'current' and all the underlying triggering events would be categorized as closed (and optionally acknowledged), and thus be mostly hidden.

Synthetic Events and Storm Control

All synthesis rules are applied independently, thus a single event could be a trigger for multiple synthetic events. This is desirable for example for detecting both per-customer problems and global issues at the same time: a few problem events can trigger a customer-specific action, while the same events could be counted together with others for detecting and reacting to a major outage.

Great care has been taken to avoid event storms caused by synthetic events: When a synthesis rule fires because there were more than `count` matching events in the time window, then all the matching events are marked as *consumed* and will not be considered for any future synthesis *for this rule.* In other words, there is no overlap between successful synthesis time windows. If a rule does *not* trigger because there are fewer than `count` trigger events, then naturally these events remain potential triggers until the time window moves past them.

However, synthetic event creation currently happens immediately as soon as a sufficient number of triggers are detected: assuming a trigger of a minimum 20 events in 60 seconds, receiving 100 events in that time frame will cause a new synthetic event for each of the 20 sufficient triggers.

Inhibiting Correlation (Version 2.2 and newer)

Version 2.2 provides a new capability for fine-tuning storm control: the `inhibit` timer.

If a correlation rule fires, and if that rule contains a numeric `inhibit` parameter greater than zero, then opEvents will temporarily disable the rule with its particular `groupby` context for that many seconds.

The primary application of this feature is to stop 'nuisance' repeat synthetics if a very large number of triggers arrives in a very short time frame: it lets you tell opEvents to generate *at most one instance* of a particular event every `inhibit` seconds.

Here is an example scenario: let's assume a rule for raising a 'Group Outage' event if 20 instances of a particular event are seen within a window of 60 seconds. A major outage happens, and 100 such trigger events for group A arrive within just a seconds, and a further 25 triggers for group B.

1. Without `inhibit`, after the first 20 events for group A you'll get one synthetic event for group A; another after the next 20 and so on.
   For group B, one synthetic event will be generated for the first 20; the remaining 5 are too few to trigger anything.
2. With `inhibit` set to 40 seconds (for example), you'll get the very first group A synthetic event as before, but then no synthetic events for this rule and group A for the next 40s;
   After that correlation for group A resumes 'from scratch' and any events received from then onwards are counted and correlated as normal.
   For group B with its fewer triggers the inhibit behaviour doesn't change anything visibly, there's still just one synthetic event.
   Note that the inhibit timer for group A is totally independent of any inhibit for group B: inhibit applies to a particular rule and its full `groupby` context.

In opEvents version 2.2, the combination of the options `autoacknowledge` and `inhibit` does *not* acknowledge trigger events that occur during the inhibit period; only 'successful' triggers are acknowledged. This has been changed for greater consistency and better storm control in versions 2.2.1 and newer, where successful triggers and any trigger events occurring during the inhibit period are also acknowledged automatically.