Understanding Orion Advanced Alerts

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orion Alerting</td>
<td>1</td>
</tr>
<tr>
<td>Orion Advanced Alerts</td>
<td>2</td>
</tr>
<tr>
<td>The Alert Process</td>
<td>2</td>
</tr>
<tr>
<td>Alert Status and Action Delays</td>
<td>3</td>
</tr>
<tr>
<td>Alert Creation, Storage and Logic</td>
<td>3</td>
</tr>
<tr>
<td>Custom Properties and Alerts</td>
<td>11</td>
</tr>
<tr>
<td>Troubleshooting Alerts and Common Alert Issues</td>
<td>14</td>
</tr>
</tbody>
</table>

This paper examines how alerts work in SolarWinds Orion NPM and related NPM modules. It also includes information on alerting logic, common issues with alerts and Orion Advanced Alert features.
Orion Alerting

Alerts perform a very important function for network managers: they allow you to instantly isolate and understand network issues as they happen, without having to watch the console screen constantly. In SolarWinds Orion products, alerting consists of a series of moving parts that can include database queries for elements out of thresholds, traps reception, syslog reception, suppression, and other qualification logic. Alerting may also include an alert action (such as an email), a reset condition, and notification, as well as a reset action. It is because of the many interactions, that alerting can be complicated. This paper is intended to examine each part of alerting, allowing for a better understanding of how alerting works and easier implementation.

Orion contains alerting capabilities for polled data, syslog and SNMP trap data. These data types each have different characteristics and each is processed by an alerting mechanism designed to best fit the data specific type. Below is a short description of each alerting service.

- The Orion Basic Alerting and Advanced Alerting programs are used to alert on polled data. Polled data is received at a predictable rate for a set number of elements Orion is monitoring. This can consist of SNMP performance data, availability and response time information, WMI application data, and other types of data residing in the Orion database.

- Syslog alerting is handled by the Orion syslog service and configured in the syslog viewer program. Syslog is a real time, push-based protocol allowing networked systems to send clear text information about their system status to a syslog receiver. Orion syslog alerting allows for alerting on syslog specific information such as a text patterns in syslog messages or messages coming from a specific host. As syslog messages are real time, Orion handles syslog alerting in real time as well, processing syslog messages as they are received and triggering configured syslog alerts and alert actions.

- SNMP trap alerting operates through the Orion SNMP Trap service, similar to the syslog viewer alerting option. Traps are realtime, push-based alerts delivered by the SNMP trap mechanism. Similar to syslog, SNMP traps are sent from networked devices and received by the trap receiver. Rather than sending text strings as syslog does, SNMP traps send SNMP MIB data, normally regarding a problem issue such as an IP interface shutting down. SNMP does this by defining certain conditions as trap Object Identifiers (OIDs) and transmitting that OID when the defined event occurs.

This paper focuses on Orion alerts and the Advanced Alert Manager with a focus on understanding how to create, use and troubleshoot advanced alerts. It is the intent of this paper to demonstrate how alerting works within Orion and not to provide a step-by-step guide for all alerting features. Features such as alert variable definitions and usage are covered in the Orion NPM Administrator Guide and will not be repeated here. Please refer to the Orion NPM Administrators Guide for directions on configuring Orion basic alerts, advanced alerts, syslog, and trap alerts as well as examples of common alerts actions. The reader should have a basic understanding of how SQL works and how to read an SQL query.
Orion Advanced Alerts

The Alert Process

A brief discussion of the components and features of Orion alerting will help set a foundation for the examples and troubleshooting discussed later in this document.

The flow depicted above occurs for both alert conditions and reset conditions. Reset conditions do not contain suppressions, so the suppression portion is skipped for reset conditions. It should be noted that the suppression condition may suppress all or only part of the triggered elements for a given alert. If it suppresses all of the triggered thresholds, then the alert condition ends there. If all or part of the triggered elements do not meet the suppression criteria, those elements become fully qualified conditions, ready for the applicable alert action. After the alert action is completed the process repeats the next time the database is scanned for the condition.
Alert Status and Action Delays

Also contained in Orion alerts are alert trigger condition timers, reset timers, and alert action timers. It is important to have a good understanding of how these timers operate in order to properly apply them. Alert condition timers create a delay period from when an alert condition is found to be true, normally for the purpose of creating a second tier of threshold for the condition. The first tier being the existence of the trigger condition, the second is the duration of the condition.

The trigger condition timer is an option when defining a trigger. When a trigger condition is detected an entry is made in the Orion database to define the state of the triggered condition. The possible statuses are:

1. Trigger pending. The condition has been met but not for the trigger delay period. This is the status of an alert with an active delay timer.
2. Triggered. This is an alert condition that was triggered and also exceeded the delay timer, or a triggered alert condition with no delay timer set.
3. Reset pending. The reset condition has been met and a reset timer is active.
4. Reset. The condition has met the criteria for reset condition and reset delay.

Advanced alerting also contains a user-settable alert action delay timer feature. If this delay is applied, the alert action will not fire until the alert action timer has expired. The alert action delay timer starts as soon as a trigger condition is detected in state 2 (triggered) and will delay the alert action for the time set, as long as the condition remains true. If the condition clears, the timer is cleared. Note that none of the above states refer to the action trigger setting. The only interaction the action delay timer has with the alert state table in the Orion database is to detect state=2 and run the timer.

Alert Creation, Storage and Logic

The Orion Advance Alert Manager provides a means to create SQL query-based alerts without having to enter queries directly into SQL. As such, the Alert Manager works as a translator, changing plain English descriptions of alert conditions into SQL and storing them as SQL queries in the Orion database.

SQL has very strict rules around defined statements or terms. These are designed to eliminate any possible vagueness or ambiguity. SQL terms are English words which, by themselves, can contain much vagueness and ambiguity. Allowing the level of ambiguity English contains in SQL would break one of the first rules of computing – “The machine should not do what you want it to do; only what you tell it to do.”

Because the Alert Manager uses plain English phrases to define alert conditions, it is crucial that you have a solid understanding of what these are and how they are defined in terms of logic. Here we will look at some of the alert options, the logic they create, and the resulting SQL. A common example of a plain English statement that is problematic logically due to ambiguity is the phrase, “All that glitters is not gold.” If we had to make rules based on this statement we could possible make the erroneous rule, “Things that glitter are not gold.” This is because the original statement can be interpreted to mean that if something glitters then it is not gold, rather that if something glitters it may or may not be gold.
Understanding Condition Group Logic

Here is a very simple alert to detect if an interface is experiencing large numbers of small packets, perhaps an indication of an attack. The trigger condition uses a condition group with an all option. This option means that all of the conditions in that group must be met for that condition group to trigger an alert.

![Edit Alert](image)

After creating an alert you can see the SQL query created for that alert in the dbo.AlertDefinitions table. The following is found in the SQL table after creating the above alert:

```sql
SELECT Interfaces.InterfaceID AS NetObjectID, Interfaces.FullName AS Name
FROM Interfaces
WHERE ((Interfaces.InPps > 10000) AND (Interfaces.Inbps < 51200))
```

The Alert Manager is translating "Trigger alerts when all of the following apply" to the defined AND Boolean. We are requiring all the conditions to be true to qualify an element as triggered. The logic diagrams for the all qualifier are shown below.

**Note:** You cannot edit the SQL statements in the dbo.AlertDefinitions NetPerfMon table. If you do, your change is overwritten. The only way to change the SQL in this table is through the Advanced Alerts feature.
The top portion is a visualization of all the applicable elements in the database being passed through the criteria of the two triggers (a and b) and the group logic (all or AND). The check marks represent where the individual condition, a and b, are true. The triggered elements at the bottom of the diagram show which elements had true trigger conditions and met the group logic. The Venn diagram is another way to visualize what will trigger the alert, only things that are both a and b. This is fairly straightforward.

There are three other options that can be used in condition groups.

Here we’ll take a look at these options and the logic they yield.

By changing the alert condition to any we get the following SQL.

```sql
SELECT Interfaces.InterfaceID AS NetObjectID, Interfaces.FullName AS Name
FROM Interfaces
WHERE ((Interfaces.InPps > 10000) OR (Interfaces.Inbps < 51200))
```
As you can see, this would be incorrect logic to trigger on the condition of an interface experiencing a large number of small packets. What it does yield is shown below.

The Boolean OR is inclusive. In common English this is sometimes referred to as an “and/or”. Logically, this means that things that are included in a or b are true, including the intersection of a and b. Using the any option in this alert the result is I would receive an alert on every interface with over 10,000 packets per second and every interface with less than 51200 inbps utilization. This demonstrated how a small change within an alert condition can have a large impact on the validity of the alert.

We have already seen that the all option is correct for this simple alert. To complete the examination of the condition group options, we will look at the remaining two options, the none and not all options without regard to how they affect the validity of this specific alert.

It should be noted that using the negative logic condition groups, none and not all, can be avoided completely by using terms such as not in the all or any condition group. It is highly recommended this approach be taken for a number of reasons.

1. It eliminates the possibility of including double negative conditions such as “alert when none of the following are not true”
2. It is simpler to troubleshoot an alert if you don’t have to switch gears in you critical thinking from positive conditions groups to negative ones.
3. The logic of a true, negative condition can be difficult to express in English. For example the song, “Yes, we have no bananas”.

The Venn diagram illustrates the inclusive OR logic. In the diagram, a and b are the two conditions, and the shaded area represents the “any” condition, which includes both a and b as well as their intersection.
Changing the condition option to *none* generates the following SQL:

```sql
SELECT Interfaces.InterfaceID AS NetObjectID, Interfaces.FullName AS Name
FROM Interfaces WHERE (NOT (Interfaces.InPps > 10000) OR NOT (Interfaces.Inbps < 51200))
```

Changing the condition option to *not all* generates the following SQL:

```sql
SELECT Interfaces.InterfaceID AS NetObjectID, Interfaces.FullName AS Name
FROM Interfaces WHERE (NOT (Interfaces.InPps > 10000) AND NOT (Interfaces.Inbps < 51200))
```

While these may queries seem to be inconsistent with the English terms *not all* and *none*, remember that the SQL query is what will actually trigger the alert, not the English definition. As long as you keep in mind the actual SQL logic associated with the *none* and *not all* options your alerts will work.

### Nested Conditions

Alerts often contain nested condition groups in order to define exactly the desired alert. Nesting is implemented by using a condition group on the top level followed by a series of other condition groups.

Three condition groups exist. Here, the order of hierarchy for the condition groups is important. The top condition group is “Trigger when all of the following apply”. What this means is all of the conditions applicable to that level must be true to trigger an alert. Lines have been added to the above image to show the three levels of condition embedding and the conditions associated with each level. For the top level AND condition to be true, the two simple conditions directly under it, the “Trigger Alert when any of the following apply” second-level embedded condition group and the bottom simple condition all must be true. For the second-level condition group, “Trigger Alert when any of the following apply” to be true at least one of the three applicable conditions must be true. These are the two simple conditions under the *any* condition group and third-level embedded condition. The third-level only requires that the two simple conditions it contains are true.
This same logic is used for any number of embedded condition groups. Here are some guidelines for creating and interpreting embedded condition groups.

1. The first condition group will be the top-level condition.
2. Conditions which apply to the top level group are indicated on the next level on indentation. The second-level conditions are indicated by the next level of indentation and so forth.
3. There may be several conditions on lower levels separating upper conditions. Always rely on the indentation of the conditions, not their specific placement top to bottom.
4. Each level of condition must be met according to the Boolean logic of that condition group level alone. For example, in the graphic shown above the top level condition group logic is only AND even though embedded conditions contain groups below contain OR logic.

**Alert Suppression and Embedded Suppression**

The suppression feature is one method of further qualifying an alert condition. Suppression works just like alert creation does, it creates an additional SQL query with the suppression logic that will suppress the configured alert in the even that the suppression condition is true. Here we’ll add a suppression condition to our alert attempting to suppress it for interfaces faster that an Ethernet interface.

Looking again at the dbo.AlertDefinitions table we see the suppression field is now populated with the following query:

```sql
SELECT Count(*) AS Supress FROM Interfaces WHERE ((Interfaces.InterfaceSpeed > 10000000))
```
And the SQL alert trigger:

```sql
SELECT Interfaces.InterfaceID AS NetObjectID, Interfaces.FullName AS Name FROM Interfaces WHERE ((Interfaces.InPps > 10000) AND (Interfaces.Inbps < 51200))
```

It would appear that the alert will fire for interfaces at or under 10 Mbps bandwidth and not fire for interfaces faster than 10Mbps. This is a common misinterpretation of suppression. The way suppression works is that if the suppression condition is true, all alerts for that trigger condition are suppressed, no matter what the alert condition contains. So, because the Orion database has an interface with a bandwidth of greater than 100 Mbps this alert will always be suppressed.

Now let's compare that example with the following:

And the resulting SQL:

```sql
SELECT Interfaces.InterfaceID AS NetObjectID, Interfaces.FullName AS Name FROM Interfaces WHERE ((Interfaces.InPps > 10000) AND (Interfaces.Inbps < 51200) AND (Interfaces.InterfaceSpeed < 1000000))
```

Here all of these conditions must be true together for the alert to fire, so it will only fire for Ethernet and slower interfaces that meet the other two trigger conditions. This is an example of embedding a suppression condition in the alert condition. You may prefer this method of adding suppression as it allows you to view the trigger condition and the qualifying suppression all in one area to apply only to the conditions you want. As the alerts you create become more complex, this can greatly assist in the troubleshooting process, should that become necessary. The use of the suppression in the suppression tab or embedded depends on the exact alert criteria and should be considered on a case-by-case basis.

Using separate alert trigger conditions and suppression conditions can also lead to trigger/suppression conflicts.
This can especially be true when the “Copy From Trigger” option is used as shown below.

There is no sure way to interpret how the user wishes the logic in the trigger condition to be applied to the suppression condition, so the trigger condition is simply copied as-is.

Copying the suppression logic without editing creates an absolute conflict between the two conditions as shown below.
This is true for any of the alert features which allow copying conditions. These include Reset, Suppress, Trigger actions and Reset features. Always use caution when copying conditions from one feature to another, and do not assume the features will interpret or edit the copied condition for you.

**Custom Properties and Alerts**

Custom properties are often used to fit the management needs of a specific environment. They allow the administrator to create and assign properties to nodes, interfaces, and volumes. Custom properties can be leveraged within advanced alerts to add environment specific alert capabilities. Here are a couple of examples of the most commonly used custom properties in alerts. These cases are derived from actual use cases.

**Case 1 – Refining Alerts**

There are several built-in methods to define which elements an alert should apply to. If the qualifiers built into Orion alerting won’t define exactly the elements or conditions of interest, a custom property can be added. In this case, the environment has several physical servers hosting several virtual servers. Both server types may be running Windows 2003 or 2008 server and can be manufactured from three different vendors. The naming convention of the servers does not indicate if a server is physical or virtual. There are also two different virtualization vendors used. The issue is that the Orion administrator needs to send an alert to the server administration department if a physical server hosting virtual servers has less than 1 GB available memory. Here is how this is solved:

1. In the Orion Custom Property (Start > All Programs > SolarWinds Orion > Grouping and Access Control > Custom Property Editor) create a new custom property based on nodes to indicate that a node is a host to virtual machines.
   
   **Note:** For more information about the Custom Property Editor, see “Creating Custom Properties” in the SolarWinds Network Performance Monitor Administrator Guide.

2. Check off the nodes that are known hosts.

3. Create the alert and include the node custom property “Is Host” = Yes

```plaintext
# Edit Alert

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Condition</th>
<th>Reset</th>
<th>Alert</th>
<th>Suppression</th>
<th>Time of Day</th>
<th>Trigger Actions</th>
<th>Reset Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Property to Monitor:</td>
<td>Node</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Trigger Alert when all of the following apply:

- Machine Type is equal to Windows 2003 Server
- Machine Type is equal to Windows 2008 Server
- Memory Available is less than or equal to 1023741824
- Is Host is equal to Yes

Do not trigger this action until condition exists for more than 0 seconds

OK Cancel Help
```
The alert is actually quite simple now and fits the use case of defining physical servers as physical servers and is independent of the server manufacturer and virtualization vendor. This allows the Orion administrator to control this qualifier. So if new vendors are used it will not affect the alert capability, as would happen if the alert was tied to a variable system qualifier.

**Case 2 – Suppressing Multiple Alerts for an Unreachable Remote Site**

Perhaps the most common use of custom properties is to create a suppression condition for devices where the reachability of those devices from Orion is dependent on the status of a single connecting device, such as a gateway router. This is common where a remote site is connected to the Orion server location via a single WAN link. There may be scores of devices located in the remote location which are managed by Orion. If the WAN link to the remote site or the gateway router at the remote site fails, then scores of alerts would be generated to indicate the down status of all the unreachable devices. By applying custom properties to the devices at the remote site and including a dependency on the remote gateway router, we can avoid the multiple alerts and only alert on the actual issue - the remote site is unreachable. For this example the remote site hosts the 10.199.4.0/24 network and the remote gateway router is at 10.199.4.1. Here is how this alert is created.

1. Create a text custom property called “Network” and place the network address 10.199.4.0/24 in the Network Custom Property for each node in that network.

2. Create an alert for nodes status not equal to up and Network equal to 10.199.4.0/24.

![Edit Alert](image-url)
3. Create a suppression condition to suppress the trigger if the remote gateway status is not equal to Up.

The choice to use “Not equal to up” instead of “Is equal to down” was made so that if there is a connectivity issue affecting the reachability of devices in the remote office and the gateway and/or the remote devices are in an unknown state, we want to suppress all those alerts also.

Now we have an alert that will let know when a node or nodes at the remote site are not up and it is not due to a failure of the WAN link to that site or the remote gateway. One issue is that if the remote gateway does fail, all of the alerts for the remote site will be suppressed, including the one for the failed gateway. To remedy this we create a second alert for the gateway router only.
Troubleshooting Alerts and Common Alert Issues

Troubleshooting alerts is very specific to the type of alert and the network environment. For this reason there is no single step-wise troubleshooting method that will expose the problems with a specific alert. There are best practices and common issues that should be considered when examining an alert problem. Here we’ll examine those practices.

Normally the way a non-functioning alert is found is due to the lack of the associated alert action occurring. There are two possible reasons for this to happen:

1. There is a problem with the alert.
2. There is a problem with the alert action.

The best place to start when an alert action fails to occur is to determine which one of these two areas is the problem.

Error logs

Error messages related to alert problems can be found in the Windows Event Log and the dbo.EventDefinitions table. These logs will tell you when an alert failed and a general reason why. This can be helpful in determining if the alert failed and is logged or if the alert action failed. If there is no error found in the log, the problem is probably with the alert action.

Test Firing an Alert

The Alert Test Fire feature can be useful if you suspect that an alert has problems that will keep an action from occurring. While this feature will test fire the alert action, keep in mind that it ignores alert suppression conditions and overall alert logic.

Manually Testing Alerts

A very specific way to test alerts is to copy the alert trigger SQL query from the dbo.AlertDefinitions table and executing it against the Orion database in SQL Server Management Studio. If there is a problem with the SQL created from the alert this will highlight the problem area. The suppression and reset SQL statements can also be tested this way.

Altering alerts to fire for a known, existing condition can be helpful in the troubleshooting process, but caution should be exercised. For example, an alert to page the SQL server team when a SQL server is running low on disk space could be considered business-critical. Testing this alert by simply setting the threshold such as “alert when available disk space is less that 99%” would be an effective test, but every server will probably qualify for this alert condition and the SQL team won’t appreciate the flood of pages. To accomplish this type of test it is a best practice to use a trigger level that will be true for one or two devices and to limit the scope of the alert actions. The testing process may include several individual tests, so limiting the scope of the test without altering the alert logic will provide a real-world test of the alert with minimal impact. Once the alert is validated at the altered thresholds and actions, the proper thresholds and actions can be restored for a final single test.

Sometimes when an alert is tested the result of the test is a completely unexpected or ludicrous result. The most common issues causing this are:

- Errors in the alert condition parameters chosen to define the condition
- Errors in the hierarchy of nested condition groups
- Errors in the selection and usage of action variables.
It can be difficult to see errors in the alert condition by looking at the Alert Manager’s alert condition screen, especially with nested conditions covering several levels and lines. The error can sometimes be better visualized by examining the trigger SQL statement copied from the dbo.AlertDefinitions Table to a text editor.

**Alert Export and Import**
Some Orion users have lab installations as well production Orion systems. This can be very valuable when testing new alerts or troubleshooting existing alerts. If an alert is problematic on the production system, the Alert Manager program can be used to export the alert to a file and then it can be imported to the lab environment to be tested and corrected if necessary.

**thwack Community**
The thwack community has thousands of users who regularly exchange ideas and solutions. Chances are that if you are trying to create a complex alert and having issues, there is a thwack user who has already solved that issue. SolarWinds employees from Development, Support and Product Management regularly interact with SolarWinds product users on thwack, so we may also be able to help you solve your problem there too.

The Content Exchange portion of thwack can be used for uploading and downloading alert templates. These templates can be imported to your Orion system and customized to fit your environment.