

# SolarWinds Technical Reference

## Understanding Orion Advanced Alerts

Orion Alerting .....	1
Orion Advanced Alerts .....	2
The Alert Process .....	2
Alert Status and Action Delays .....	3
Alert Creation, Storage, and Logic.....	3
Understanding Condition Group Logic.....	4
Nested Conditions and Condition Groups .....	7
Explicit Alert Suppression and Embedded Suppression .....	8
Using Dynamic Service Groups to Simplify Alerts .....	12
Using Dynamic Service Groups and Dependencies to Suppress Multiple Alerts .....	14
Custom Properties and Alerts .....	15
Troubleshooting Alerts and Common Alert Issues.....	17

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 as of NPM 10.1.

Copyright© 1995-2011 SolarWinds. All rights reserved worldwide. No part of this document may be reproduced by any means nor modified, decompiled, disassembled, published or distributed, in whole or in part, or translated to any electronic medium or other means without the written consent of SolarWinds. All right, title and interest in and to the software and documentation are and shall remain the exclusive property of SolarWinds and its licensors. SolarWinds Orion™, SolarWinds Cirrus™, and SolarWinds Toolset™ are trademarks of SolarWinds and SolarWinds.net® and the SolarWinds logo are registered trademarks of SolarWinds All other trademarks contained in this document and in the Software are the property of their respective owners.

SOLARWINDS DISCLAIMS ALL WARRANTIES, CONDITIONS OR OTHER TERMS, EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, ON SOFTWARE AND DOCUMENTATION FURNISHED HEREUNDER INCLUDING WITHOUT LIMITATION THE WARRANTIES OF DESIGN, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL SOLARWINDS, ITS SUPPLIERS OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, WHETHER ARISING IN TORT, CONTRACT OR ANY OTHER LEGAL THEORY EVEN IF SOLARWINDS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Document Revised: 2/21/2011

## Orion Alerting

Alerts perform a very important function for network managers. They allow you to instantly isolate and understand network issues as they occur, without having to constantly monitor the console screen. In SolarWinds Orion products, alerting consists of a series of components that use database queries to identify elements outside their threshold limits, trap reception, syslog reception, suppression, and other qualification logic. Alerting may also include an alert action (such as an email), a reset condition, as well as a reset action. It is because of the many interactions involved in this process that alerting quickly becomes complicated. This paper is intended to examine each component of alerting, providing 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 type. Below is a short description of each Orion alerting service.

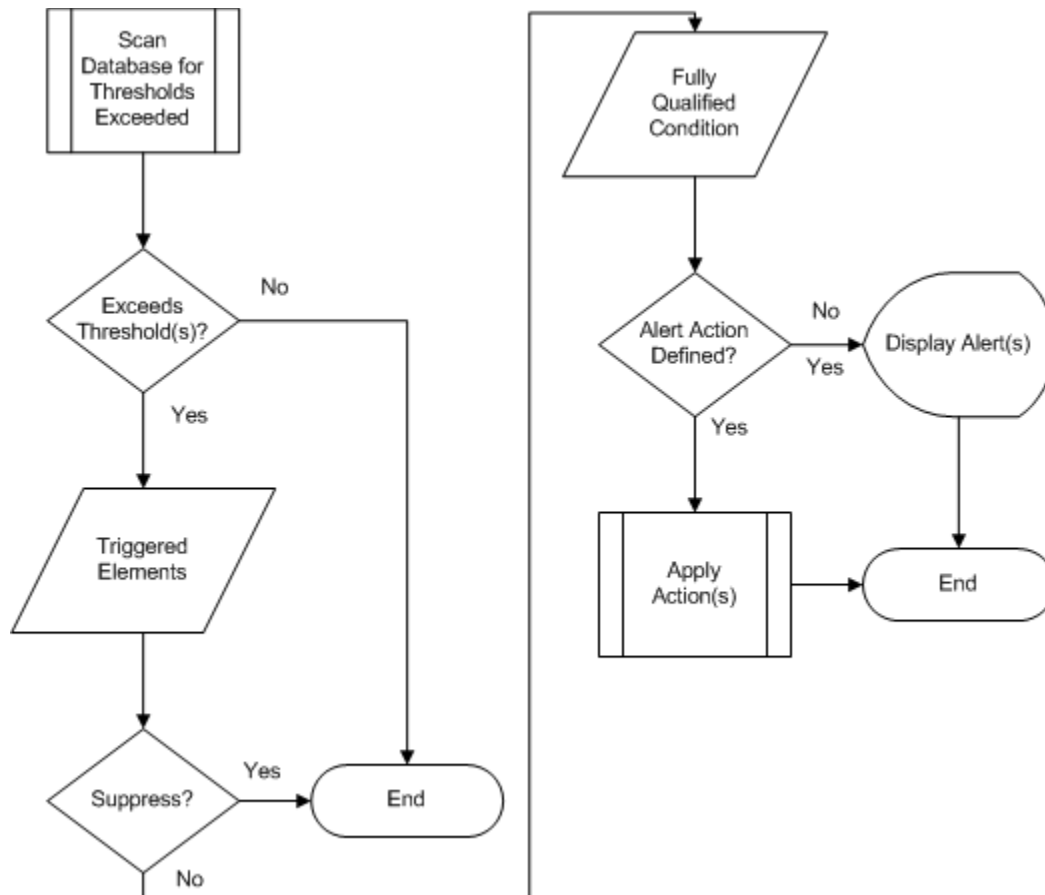
- The Orion Basic Alerting and Advanced Alerting Manager programs are used to alert on polled data. Polled data is received as requested for a set number of elements Orion is monitoring. This can consist of SNMP performance data, availability and response time information, WMI application data, and other types of data.
- 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 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 real-time, 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 an issue such as an IP interface shutting down. SNMP does this by defining certain conditions as trap Object Identifiers (OIDs) and transmitting the proper trap OID when a defined event occurs.

This paper focuses on Orion Advanced Alert Manager with an emphasis 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 Administrator Guide* and will not be repeated here. Please refer to the *Orion Administrator Guide* and the *Orion Administrator Guide* for directions on configuring Orion basic, advance, syslog, and trap alerts as well as examples of the most common alerts. The reader should have a basic understanding of how SQL works and how to read a simple 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 shown above occurs for both alert trigger conditions and reset trigger conditions. Reset conditions do not contain suppressions, so the suppression portion is skipped for resets. 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 alert conditions.

## ***Alert Status and Action Delays***

Contained within 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 qualifier for the alert condition. The first qualifier 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 configurable alert action delay timer. 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 triggered and will delay the alert action for the time set, as long as the triggering 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 alert is triggered and run the timer.

## ***Alert Creation, Storage, and Logic***

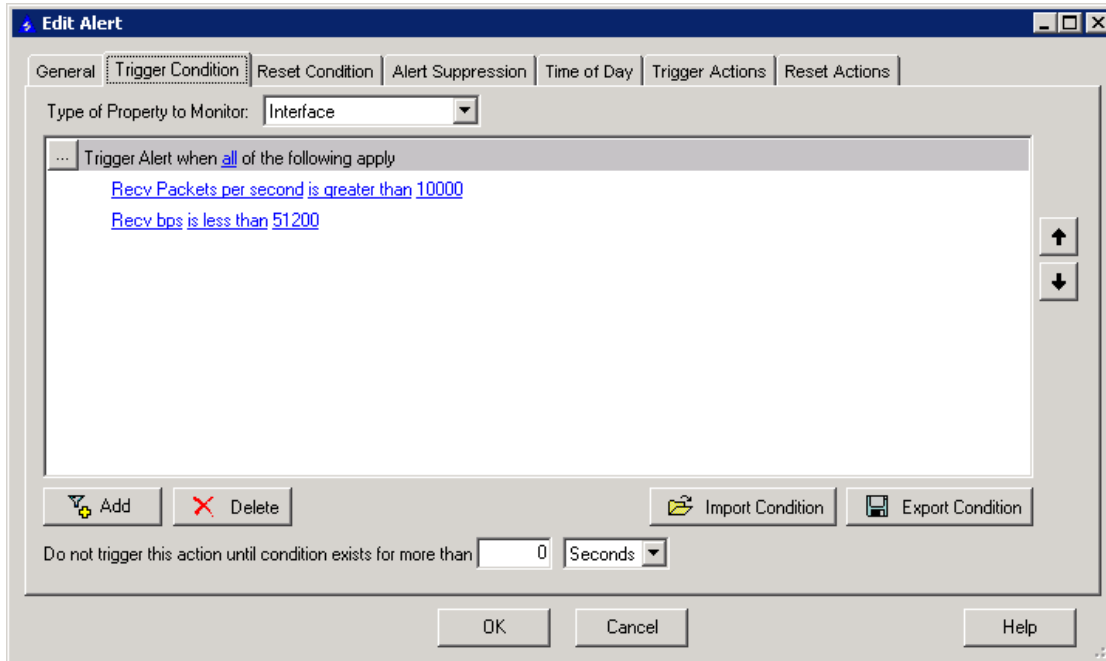
The Orion Advance Alert Manager provides a means to create SQL query-based alerts without having to enter queries directly into SQL Server Management Studio. As such, the Alert Manager works as a translator, changing plain English descriptions of alert conditions into SQL and storing them as SQL queries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

Because the Alert Manager uses plain English terms 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 SQL logic. Here we will look at some of the alert options, the logic they create, and the resulting SQL.

An 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 possibly 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.

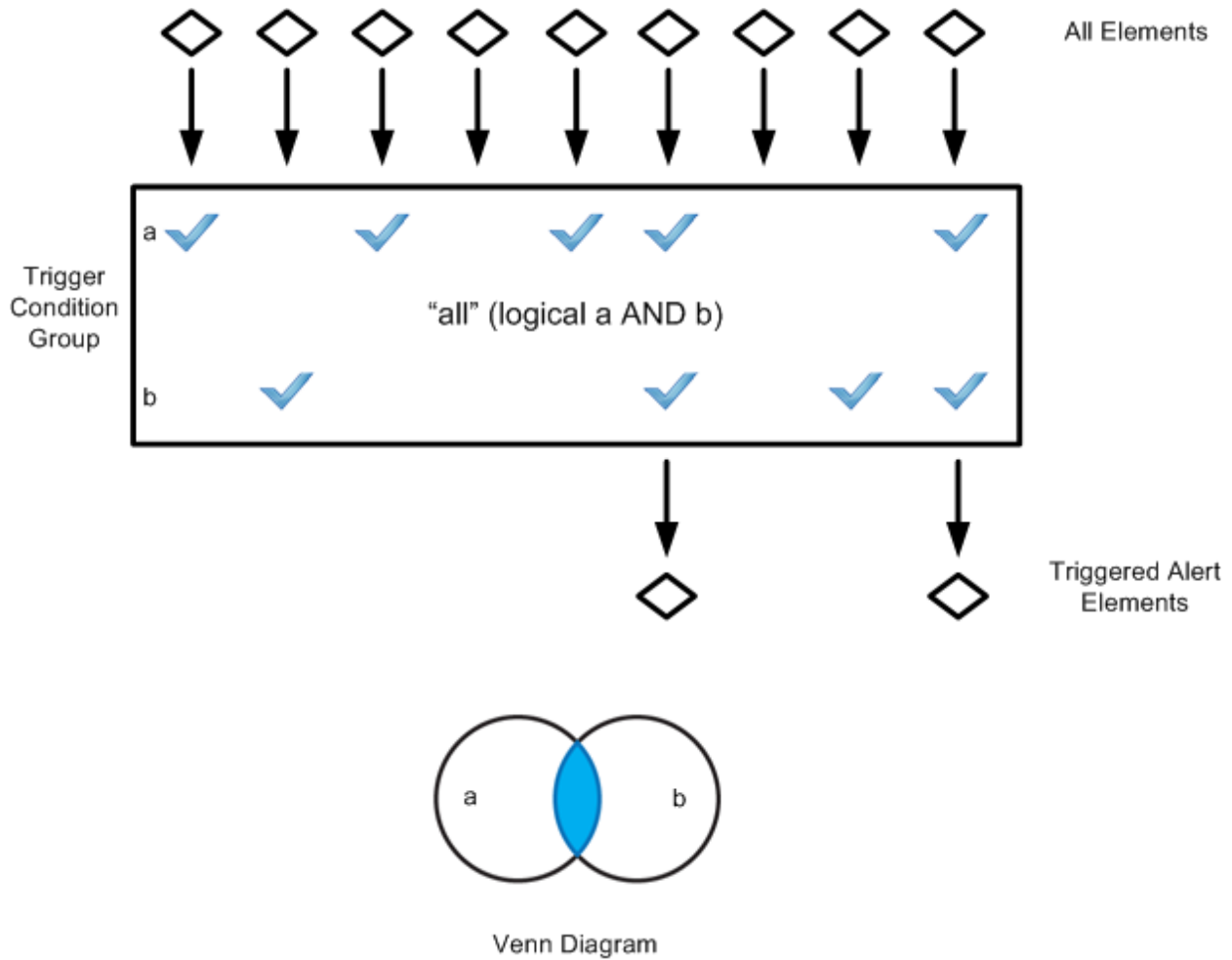


**Note:** You cannot edit the SQL statements in the `dbo.AlertDefinitions NetPerfMon` table. If you do, your change is overwritten. The only way to change the SQL query in this table is by modifying the alert in Advanced Alerts Manager.

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:

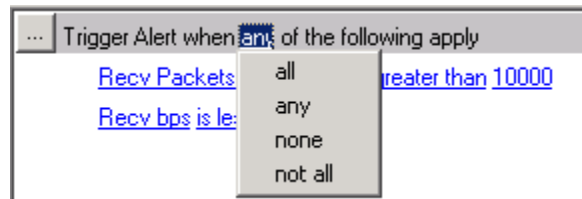
```
SELECT Interfaces.InterfaceID AS NetObjectID, Interfaces.FullName AS Name
FROM Interfaces WHERE ((Interfaces.InPps > 10000) AND (Interfaces.Inbps < 51200))
```

The Alert Manager is translating “Trigger alerts when *all* of the following apply” to the shown ‘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.



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 straight forward.

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

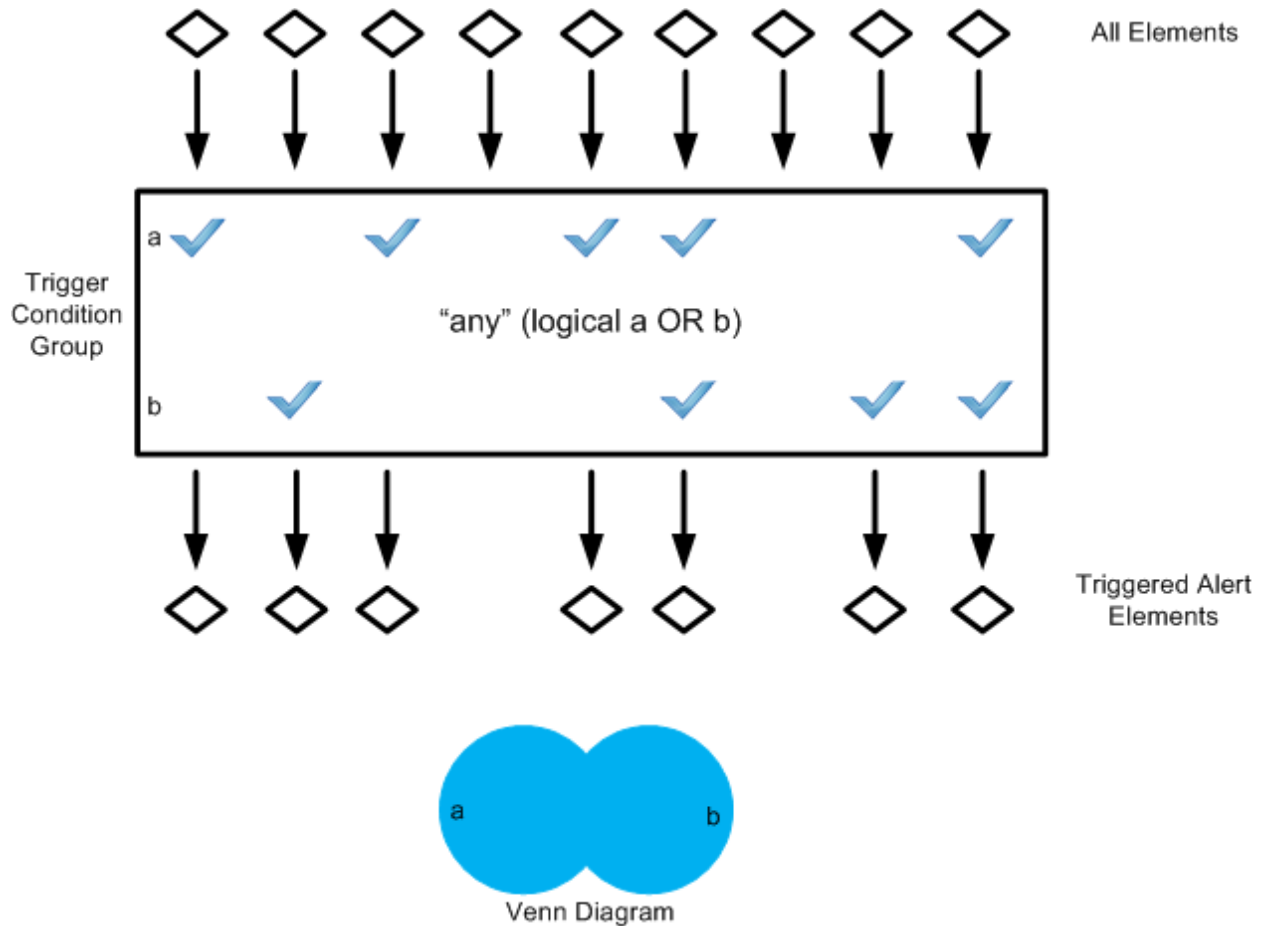


Here we'll take a look at these options and the logic they yield.

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

```
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 in bps utilization. This demonstrated how a small change within an alert condition can have a large impact on the behavior 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 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".

Changing the condition option to *none* generates the following SQL:

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

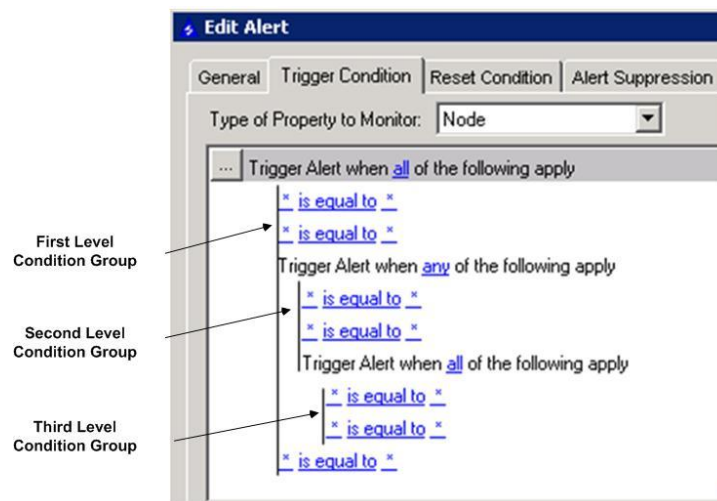
Changing the condition option to *not all* generates the following SQL:

```
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 and Condition Groups

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. The hierarchy for the condition groups is crucial to how they will function. The first level 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. Note that the first level condition group includes the first two simple conditions, the second level condition group and the last simple condition. This is shown by the vertical line added to the image on the left edge of each first level condition.

The second level condition group contains only two simple conditions under the *any* condition group, and third level condition. The third level only requires that both the two simple conditions it contains are true.

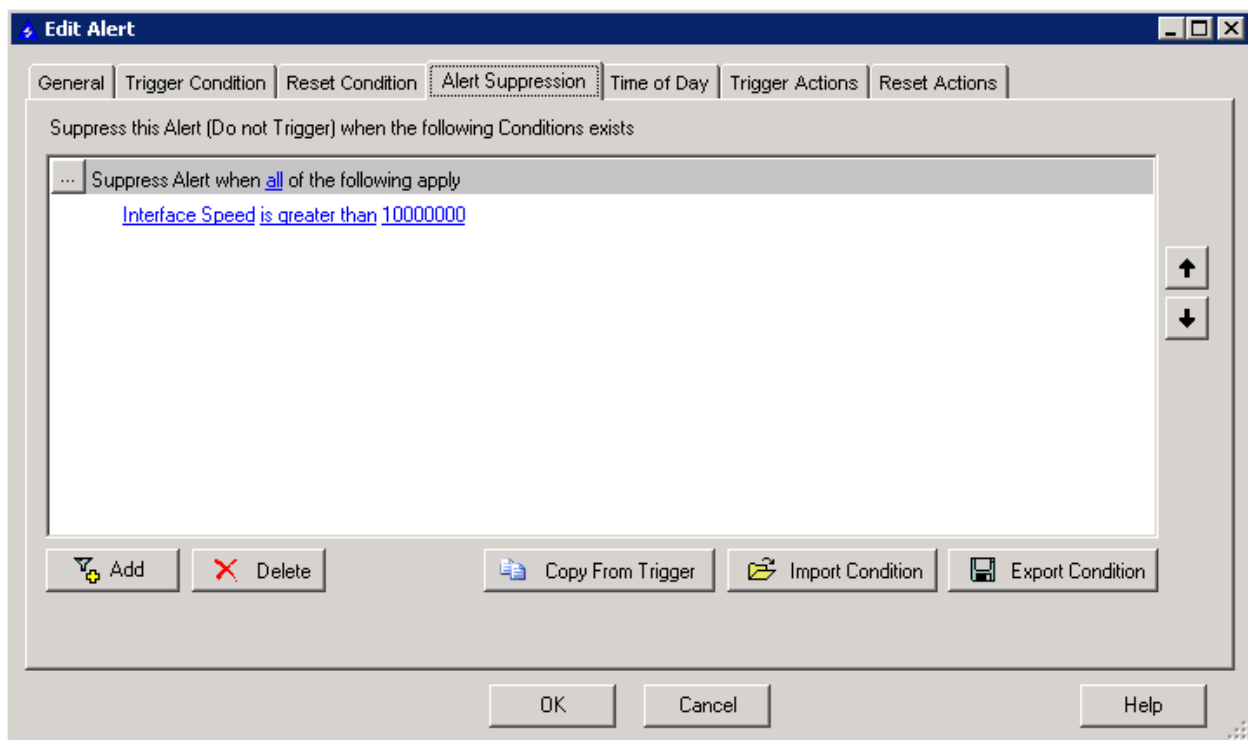
This same logic is used for any number of embedded condition groups. Here are some guidelines for creating and interpreting embedded condition groups.

1. The first condition (or condition group) will be the top line of the alert. This condition or group *must be true* for the alert to trigger.
2. For the first level group to be true, the requirements of all embedded conditions or embedded condition groups must be met.
3. Conditions which apply to the first level group are indicated on the first level of indentation. The second-level condition will always be a part of the first level conditions, and so forth.

4. There may be several embedded conditions on lower levels separating upper level conditions. Always rely on the indentation of the conditions, not their specific placement top to bottom. This is seen above as the last condition top to bottom is actually part of the first lever condition group.
5. 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 AND (all) even though embedded conditions contain groups below contain OR (any) logic.

## ***Explicit Alert Suppression and Embedded Suppression***

The Alert Suppression tab creates an explicit suppression and is one method of further qualifying an alert condition. Suppression works just like alert creation does, it creates a SQL query with the suppression logic. Here we'll add a suppression condition in the Alert Suppression tab, attempting to qualify the previous alert trigger only for interfaces faster than an Ethernet interface.



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

```
SELECT Count(*) AS Supress FROM Interfaces WHERE ((Interfaces.InterfaceSpeed > 10000000))
```

And the SQL alert trigger (as shown on page 4)

```
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 contains an interface with a bandwidth of greater than 100 Mbps this alert will always be suppressed.

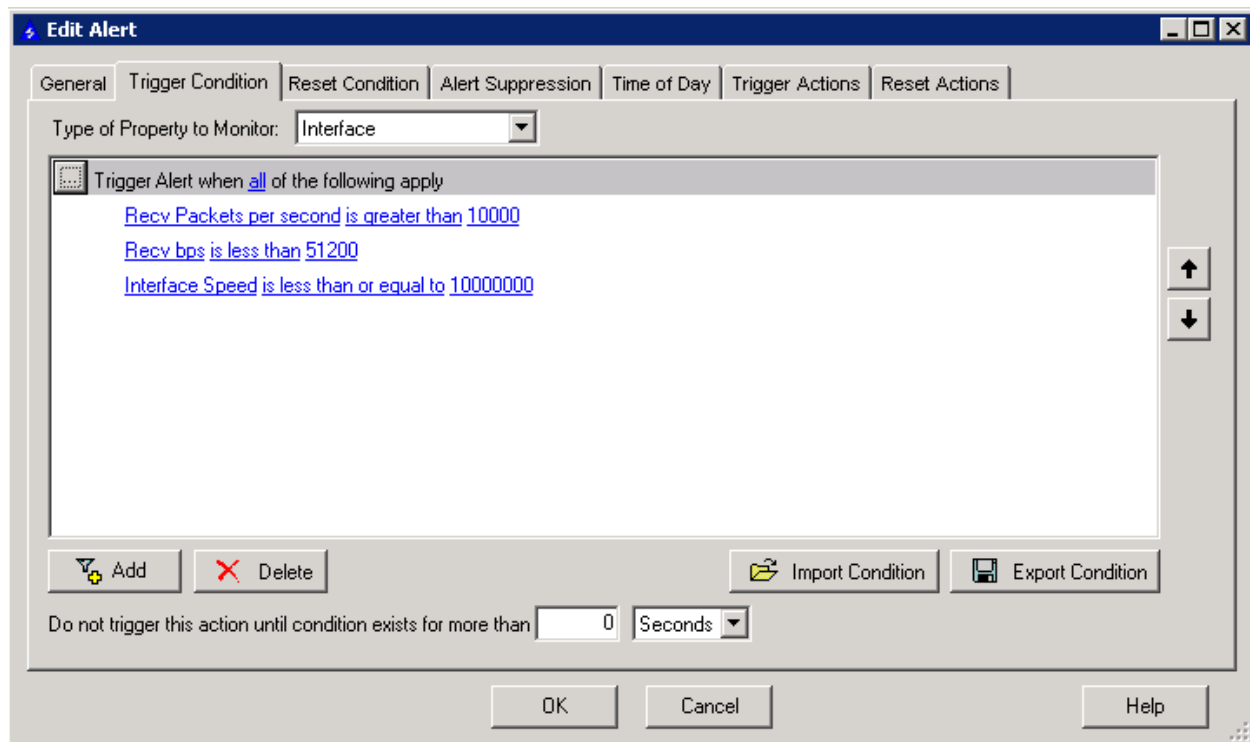
This is often seen when a user is attempting to create an alert but exclude certain devices from the alert. Typically that suppression would have the condition:

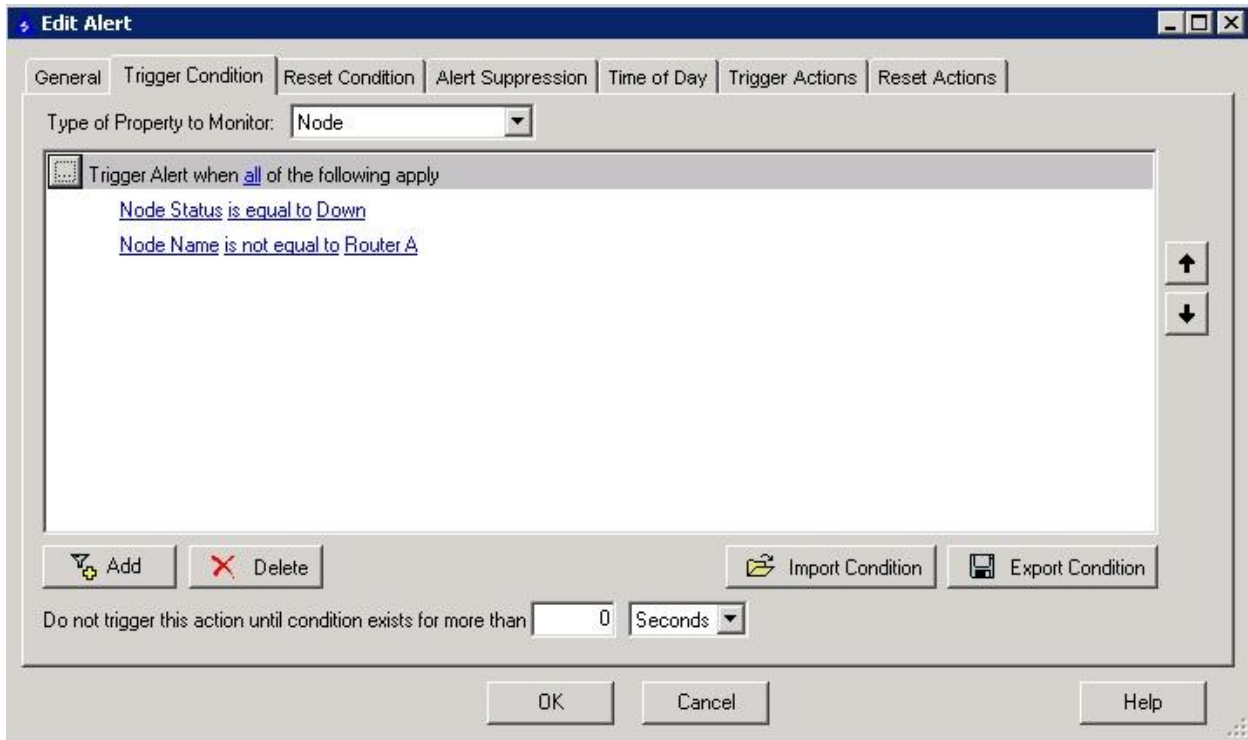
Suppress Alert when all of the following apply

Node name equal to Router A

The user is attempting to exclude router A only from the trigger alert trigger, but again, this suppression will take effect if there is a node named Router A in the Orion database regardless of the trigger conditions.

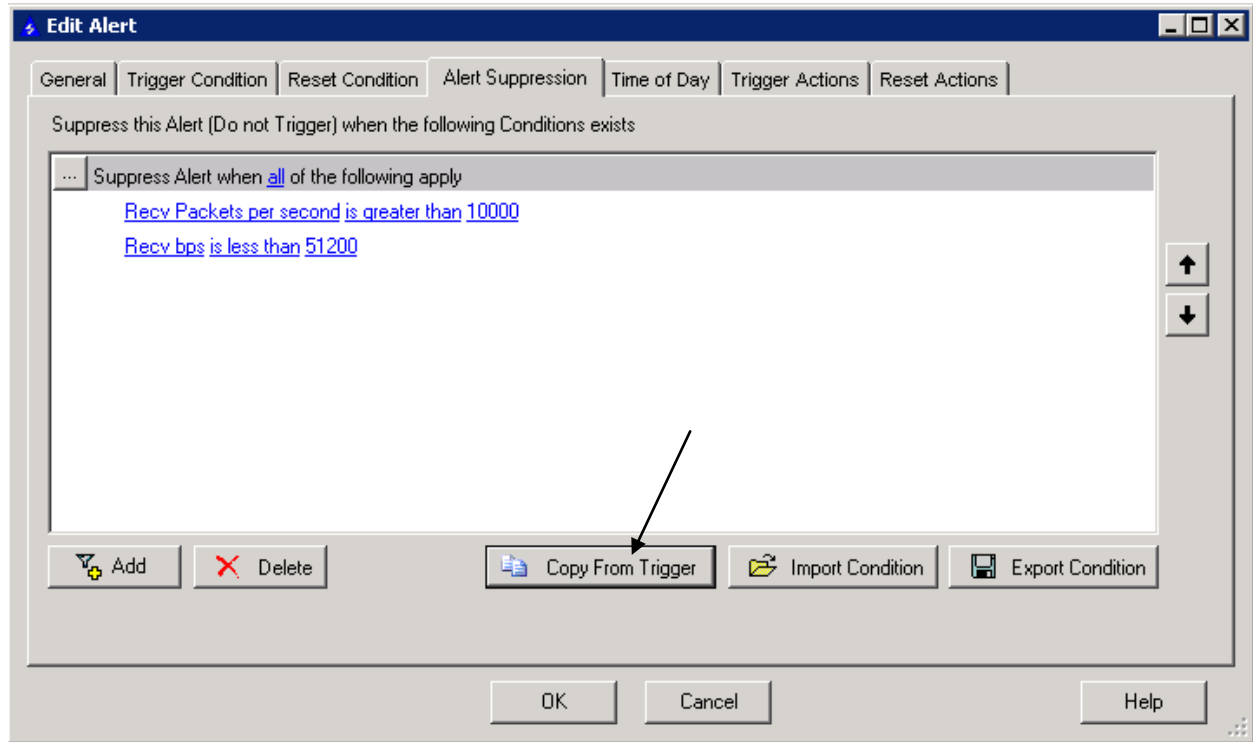
Suppression should not be used to attempt to qualify the members of an alert event. That type of qualification should be embedded in the trigger condition examples shown below.





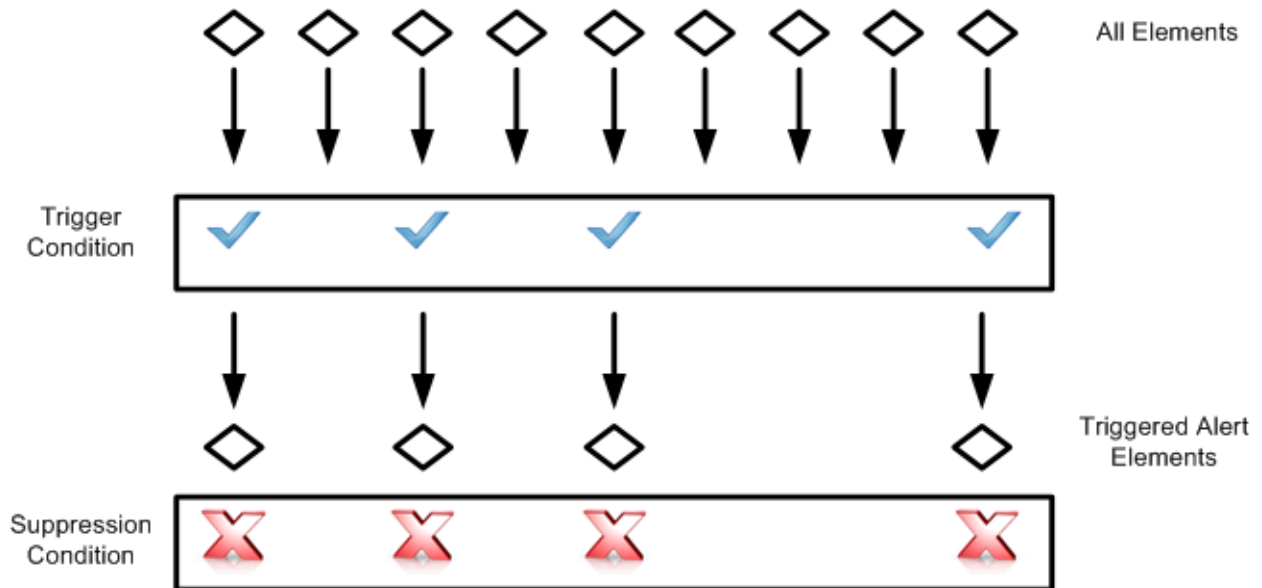
Here all of these conditions must be true together for the alert to fire. This is an example of embedding suppression conditions in the alert condition. You may prefer this method of adding suppression as it allows you to view the trigger condition and the qualifying suppression all in one area to apply 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 suppression depends on the exact alert criteria and should be considered on a case-by-case basis.

Using separate alert trigger conditions and explicit suppression conditions may 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, including 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. The explicit suppression feature is best used to associate the state of two or more conditions, rather than qualifying members of a condition. Suppression can be used to eliminate multiple alerts for a situation where a group of managed nodes becomes unreachable due to the failure of the connection between NPM and the managed objects. The Service Group feature added in NPM 10.1 addresses this need to better associate dependent devices with the device or path they depend on.

## Using Service Groups to Simplify Alerts

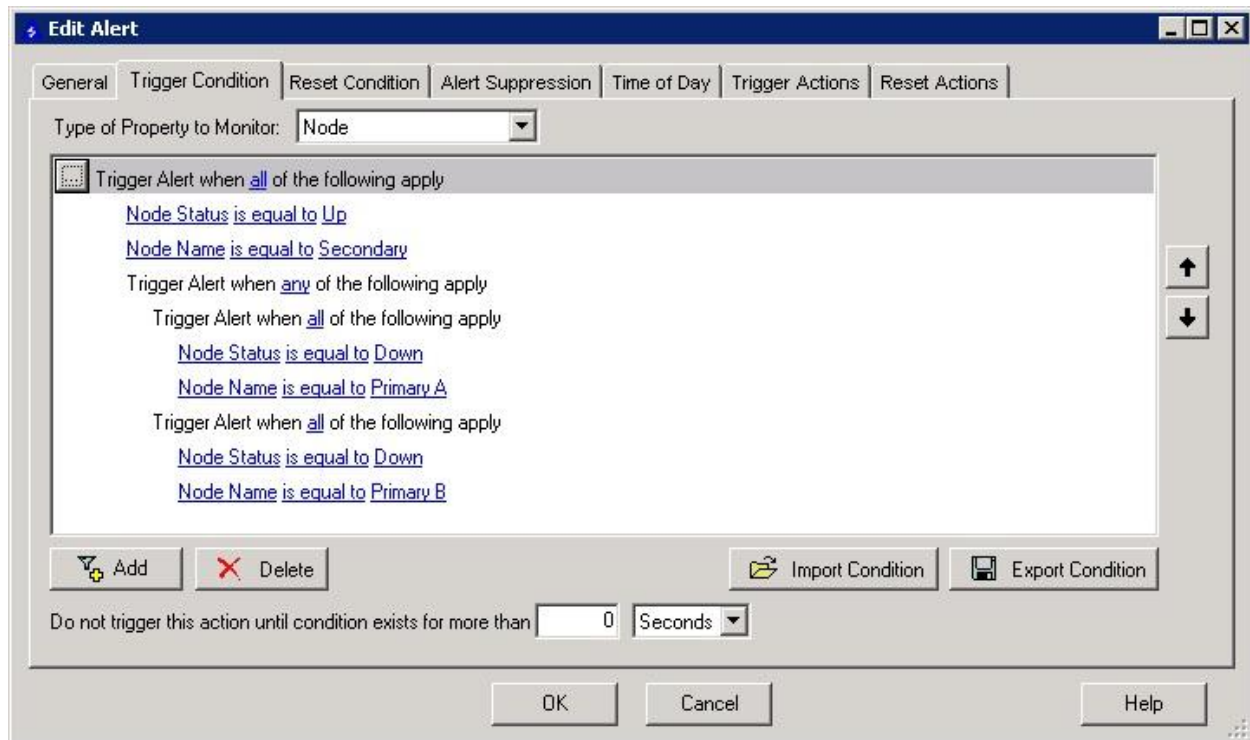
In this example an alert is required to indicate that any of several primary devices has failed and a specific secondary device is active.

There are several methods that will create valid alerts. The challenge is to create an alert that is simple to understand, implement, and alter. At first one might consider placing all the criteria under one any (OR) condition group, such as seen below.

Trigger Alert when any of the following apply

```
Device Secondary is up
Device Primary B is down
Device Primary C is down
Device Primary D is down...
```

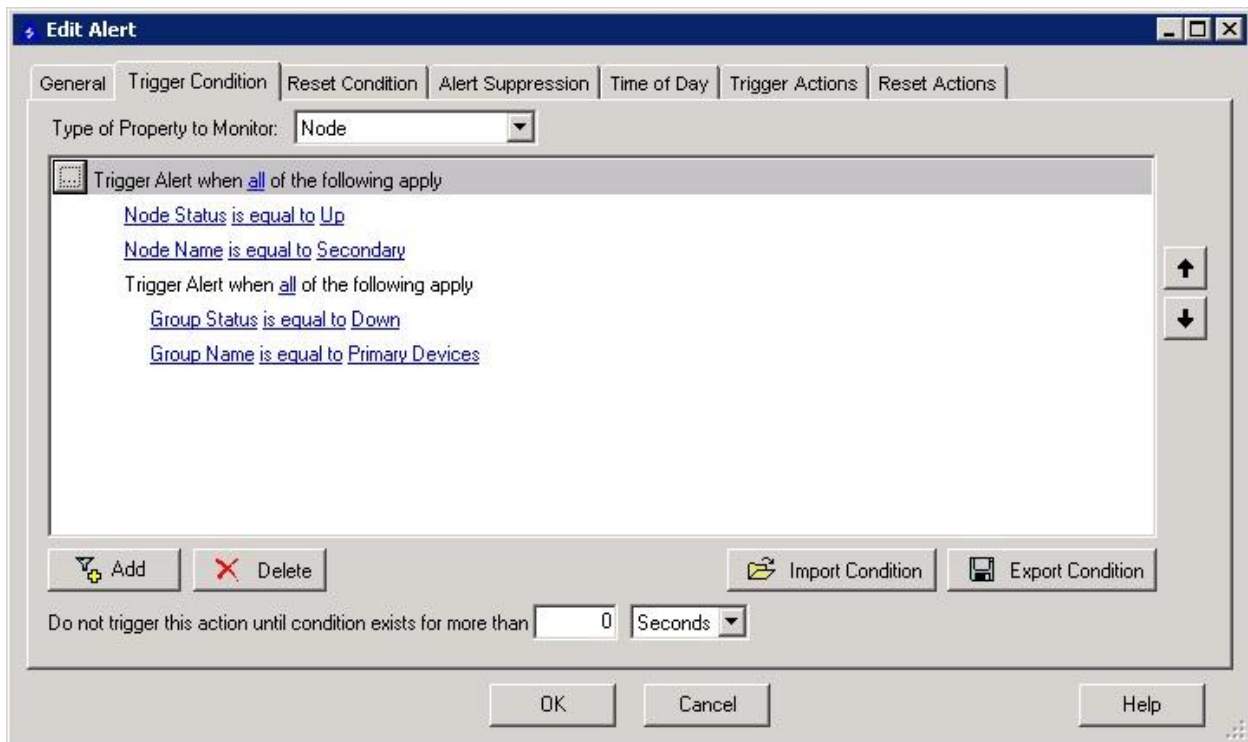
The problem here is that the alert will fire when the secondary device is up and there are no primary device failures. In fact, the logic is so flawed in this example that it is impossible to create this alert in Advanced Alert Manager. What is needed is some logic that will separate the status of the primary devices from the secondary devices, yet relate the status in a way that meets the alert goal. One possibility is shown below.



This alert has three embedded condition levels. Keeping in mind that for an alert to fire, the first level condition group must be true and all condition groups are dependent on the embedded conditions or condition groups they contain. Here is the alert logic:

1. The third level consists of two independent condition groups, one for the status of Primary A and one for the status of Primary B.
2. The second level condition group uses the any (OR) qualifier so that if any or both of the third level conditions are true, then the second level is true.
3. The first level condition group uses the all (AND) qualifier. For the first level to be true the Secondary must be up and one of the second level must be true.

Service groups were added to NPM in 10.1 and allow administrators to create groups of managed objects. These groups make the administration of several tasks in NPM much easier and add functionality not previously possible. Here, we will examine how groups can be used to simplify an alert. For step-by-step instructions on implementing and using service groups see the *Orion Administrator Guide*. One reason to create a service group is to associate like objects together so that managing them can be simplified. Compare the above alert with the one below using the service group “Primary Devices”.



Note the simplicity of this alert and yet it accomplishes the same goals as the example above it. Here is how the service group was applied to this alert.

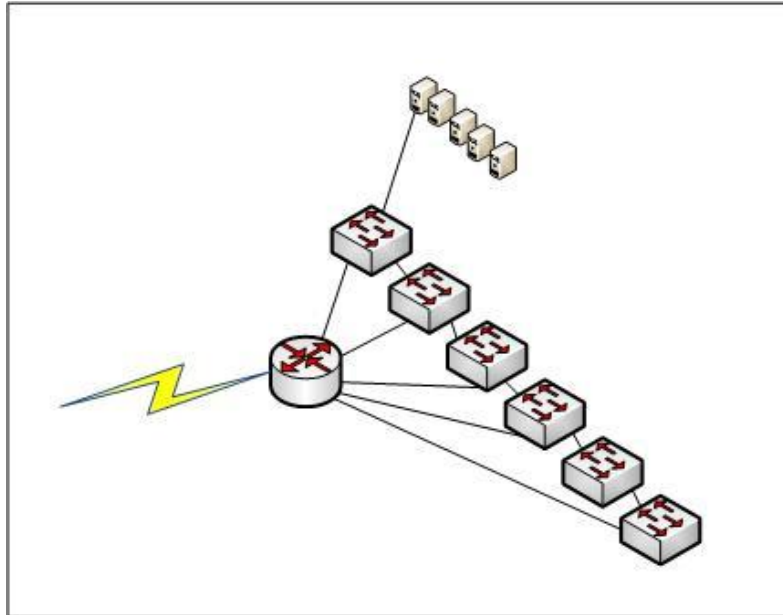
1. A new service group was created in the NPM Setting page under Manage Groups.
2. The primary devices of interest were added to the group.
3. The group status rollup mode was set at “show worst status”. This will cause the group status to show as down if any of the group members (primary devices) are down.
4. The group conditions were added to the edited alert using the Type of Property to Monitor set to “Group”.

This group may also be set to dynamically add new devices with “Primary” as a part of the device name, automatically adding them to the group and thus to the alert. This is called a Dynamic Service Group.

## Using Service Groups and Dependencies to Suppress Multiple Alerts

This section discusses the use of service groups and dependencies. For detailed instructions on implementing service groups and dependencies see the *Orion Common Components Administrators Guide*.

Consider the below diagram of a remote site.

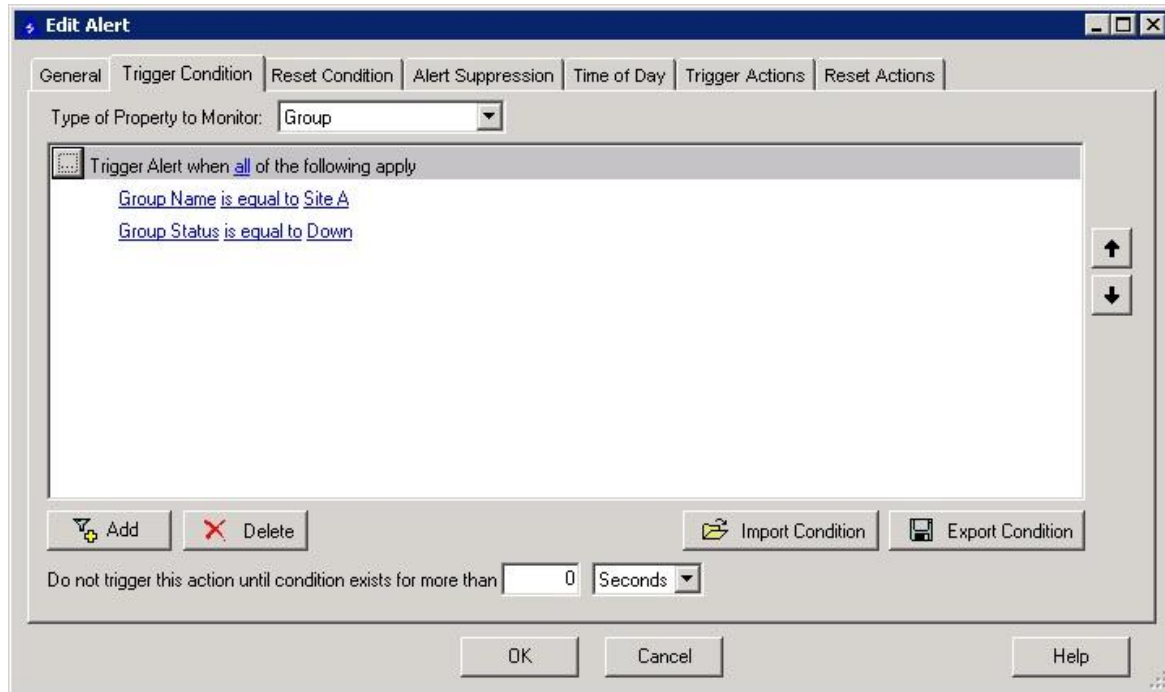


Remote Site A

This remote site and all the devices within the site are managed from an NPM installation at the headquarter site. The site is connected to the headquarters via the single WAN link shown. If the connectivity to the remote site A is severed, ICMP status tests from the headquarters to any device at site A will timeout, causing device statuses at site A to transition from “Up” to “Down”. This will cause down alerts for all the devices in site A, when the actual cause is only a WAN connectivity issue. The resulting flood of alerts could mask the valid alert that points to the WAN failure. Using service groups and dependencies allows for the creation of parent-child definitions between devices and groups or even between groups and other groups. For example, in the above diagram the status of devices within site A, as seen from outside the remote site, is necessarily dependent on the WAN link connecting the remote site to the headquarters. If the link fails, it will appear to the NPM at the headquarters that all of the site A devices have failed. Here is one way to create groups and dependencies so that the problem with WAN connectivity to site A is properly identified and superfluous alerts are eliminated.

1. Create a service group for all of the devices in remote site A including the WAN router.
2. Using the dependencies settings, set the status of the site A service group to be dependent on the status of the site A router status.
3. Create an alert for the Status of the site A service group equal to down.

Following these steps will create the site A child service group. The result will be that if NPM detects one or more of the site A devices is not reachable, NPM will check the status on the parent object, the site 1 router, to determine if the outage is related to a reachability issue. If the WAN router is not available the child devices' status change from "Up" to "Unreachable", thus not triggering "Down" alerts for the dependent devices. If, on the other hand, NPM sees that the site A WAN router is still reachable, NPM will create alerts for any site A devices with a down status. To include groups in alerts, the Type of Property to Monitor drop down box now contains properties for Group and Group Member. Using the Group Status option from the Group dropdown the alert for this service group is shown below.



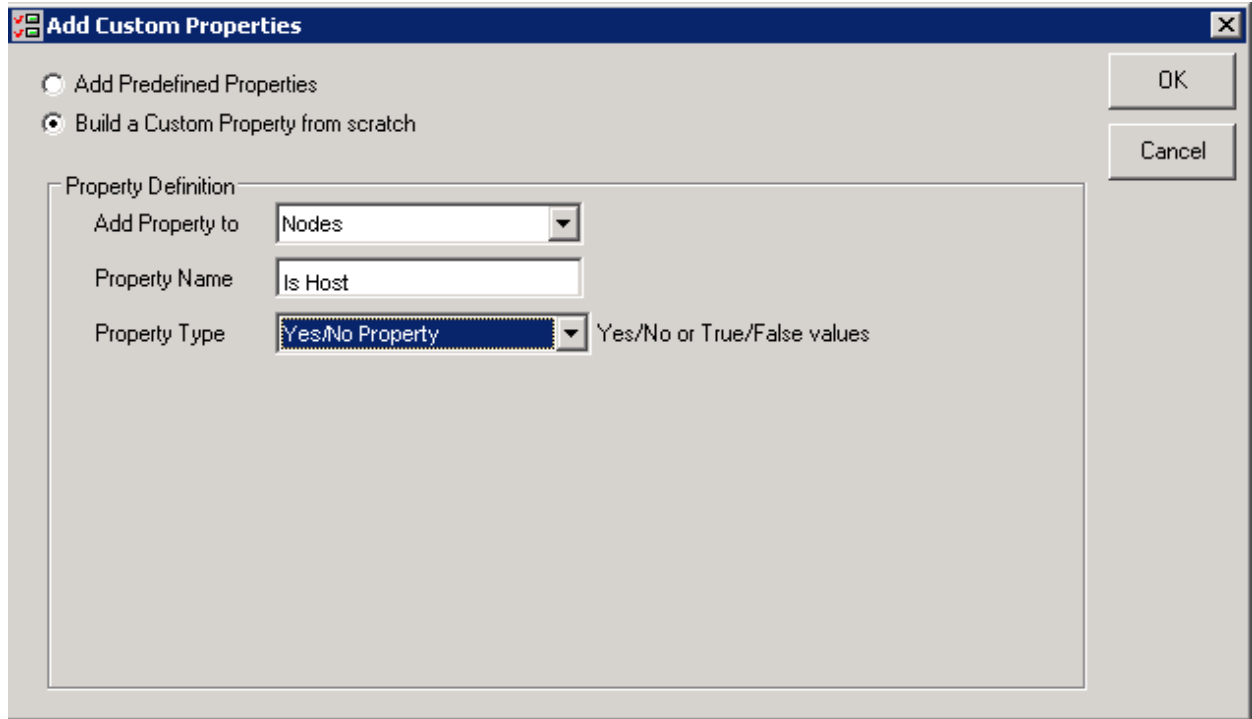
The cause of the outage is the WAN connectivity, so that should be expressed in the trigger action.

## ***Custom Properties and Alerts***

There are several built-in methods to define which elements an alert should apply to. If the qualifiers built into Orion alerting do not suffice to define the exact elements or conditions of interest, a custom property can be added. In this example, the environment has several physical servers hosting several virtual servers. Both server types may be running Windows 2003 or 2008 server and can be 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:

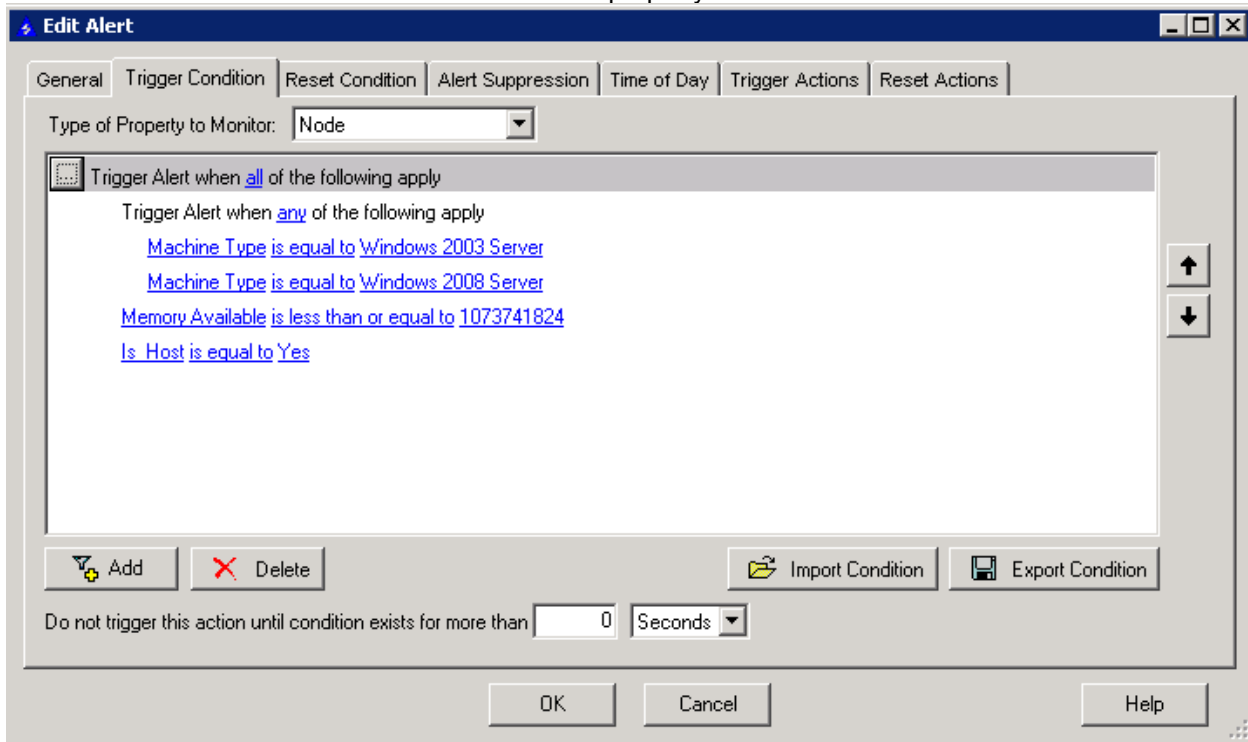
1. In the Orion Custom Property create a new custom property to indicate a node is a host to virtual machines.



2. Check off the nodes that are known hosts.

Custom Properties for Nodes		
IP_Address	Caption	Is_Host
10.199.1.1	10.199.1.1	<input checked="" type="checkbox"/>
10.199.1.107	10.199.1.107	<input type="checkbox"/>
10.199.1.11	10.199.1.11	<input checked="" type="checkbox"/>
10.199.1.115	10.199.1.115	<input checked="" type="checkbox"/>
10.199.1.117	10.199.1.117	<input type="checkbox"/>
10.199.1.118	10.199.1.118	<input checked="" type="checkbox"/>
10.199.1.121	10.199.1.121	<input checked="" type="checkbox"/>
10.199.1.123	10.199.1.123	<input checked="" type="checkbox"/>
10.199.1.140	10.199.1.140	<input type="checkbox"/>
10.199.1.174	10.199.1.174	<input type="checkbox"/>
10.199.1.178	10.199.1.178	<input type="checkbox"/>
10.199.1.181	10.199.1.181	<input checked="" type="checkbox"/>
10.199.1.190	10.199.1.190	<input checked="" type="checkbox"/>
10.199.1.195	10.199.1.195	<input checked="" type="checkbox"/>
10.199.1.200	10.199.1.200	<input type="checkbox"/>

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



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 also allows the Orion administrator to control this qualifier. 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.

## ***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 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 will 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. It is simply a test of the alert actions.

## *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 than 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.