SolarWinds Technical Reference
Using Orion Groups and Dependencies

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction – The Need to Manage Networks to Meet Business Goals</td>
<td>3</td>
</tr>
<tr>
<td>Orion Service Groups (Groups)</td>
<td>3</td>
</tr>
<tr>
<td>Nested Groups</td>
<td>4</td>
</tr>
<tr>
<td>Group Status</td>
<td>5</td>
</tr>
<tr>
<td>Dependencies</td>
<td>9</td>
</tr>
<tr>
<td>Alerting and Reporting</td>
<td>11</td>
</tr>
</tbody>
</table>

This Technical Reference demonstrates how groups and dependencies are created and provides implementation examples.

This PDF is no longer being maintained. Search the SolarWinds Success Center for more information.
Introduction – The Need to Manage Networks to Meet Business Goals

This paper will demonstrate how Service Groups and related features can be used in SolarWinds Orion products to simplify network and systems management. The reader should be familiar with SolarWinds Network Performance Monitor (NPM), Application Performance Monitor (APM), IP SLA Manager (IP SLA Manager), and the use of the Settings section of the Web Console.

At the most basic level, network management involves monitoring network assets for metrics such as performance, availability, resource usage, configuration, and appropriate usage. For the smallest of networks, this level of management works well. As a network grows in size and complexity, new management needs arise. Perhaps a small network originally designed to support email and web access is now being used to sell items on the Internet. Customer-order web servers are added along with client, inventory, pricing, and fulfillment databases. As this business grows, new systems are added and the complexity of the network increases. When a network issue occurs, it could affect any portion or function of the network. For example, let’s say a customer submits a large order, but the fulfillment database has a full drive and cannot accept the order. The customer receives an error message after filling out two pages of order and billing information, abandons the site and buys elsewhere.

The person managing this system may notice that a drive has no available free space, and replace the drive with a larger one. Between the first failed customer order and the placement of the new drive, there was some unknown number of failed orders and customers who abandoned the site. There are two glaring issues in the scenario. First, the person managing the system is only applying break/fix management and has no insight into issues before something breaks. Secondly, this person has no insight into how these individual machines are working together to support the business process.

By adding a simple Network Management System (NMS), the network manager is able to see problems develop before they become outages and quickly determine what part of this simple network needs attention. As this enterprise continues to grow, each system becomes more complex and more reliant on other systems. While the NMS implemented can scale well, it does not have the ability to manage the network in a way that is congruent with business processes and business goals. In this complex network environment complex dependencies exist where individual systems and groups of systems are reliant on other systems or groups. Managing the network as a collection of separate entities is no longer sufficient.

Orion Service Groups (Groups)

SolarWinds has introduced service groups to address the need to associate managed objects. Group objects can be physical items such as interfaces and nodes as well as logical objects such as application performance metrics or IP SLA test results. Two types of service groups can be created: static groups and dynamic groups. In static groups, all group members are manually added and the group members are only edited by manual means. In dynamic groups, the members are specified by indicating a requirement, so that any managed object that meets the requirement is added to the group. Qualifying new objects added after the group has been established will also be automatically added to the group. Likewise, objects which no longer meet the group membership are automatically removed from the group. This may occur if you edit the group requirements, or edit the properties of a group member.
Nested Groups

Groups can contain groups as members. These are called nested subgroups. Creating groups with nested subgroups can be a complicated task if not properly planned. Planning and some insight into the behavior of the Manage Groups interface can help speed the creation of groups and minimize troubleshooting. For example, let’s say I want to add all devices in a remote site called building 7, but I also want to group these devices by floor. I could create a group for bldg 7 and assign all devices in bldg 7 to the group, perhaps by using a dynamic query returning the Location field.

This will add all managed nodes that have the Location in their config set to Bldg 7. The location OID in MIB2 is a field Orion automatically polls and stores for all managed objects. This will work if all of the managed nodes in building 7 have exactly the same OID value of Bldg 7. If some were added with Build 7 or Bld 7 or a number of other nonstandard location indicators, they will be omitted. While dynamic queries can be a powerful method of managing group membership, they can cause great trouble if misapplied.

Let’s examine some more information about building 7. We find in the IP scheme that building 7 was designed to have one 10.7.X /24 subnet for each of its floors. The first floor is 10.7.1.0 /24, the second is 10.7.2.0 /24 and the third is 10.7.3.0 /24. This IP scheme is enforced in the routing protocol and by ACLs. With this information we can create the groups for building 7 and all the subgroups quite easily. Here are the steps:

1. Create an empty group called Bldg 7.
2. With the Bldg 7 group checked add a group called Bldg 7 1st floor.
3. Create a dynamic query for the Bldg 7 1st floor using IP Address begins with 10.7.1. (Be sure to use the trailing dot, otherwise the query is equally valid for .10 and .100, which are not valid first floor systems according to our schema)
4. Repeat steps 2 and 3 for all other floors, changing the group name and IP address query to match the floor number.
Once all the above steps are complete, we have created a group for building 7 containing a subgroup for each floor. There is no need to add each individual node directly to the Bldg 7 group as they are all members of Bldg 7 by being members of one of the subgroups. This can be helpful in minimizing duplicate alerts and duplicate report entities and saves the time that would be used to add items to the Bldg 7 group and then to the proper subgroup.

**Group Status**

When a group is created, you can set the status rollup mode for the group. The rollup modes allow you to make choices on how the groups status will be determined, using the status of the group members. The choices are Best status, Worst status and Mixed status. Best status shows the group status the same as the best status of any group member(s), disregarding the status of all other members. Worst status shows the group status as the worst status of any member(s), again disregarding the status of all other members. Mixed status will show the group as a single status when all members are in that particular status and warning when there are members with different status. Here are some conclusions you can make according to the status rollup you choose.

- **For Best status rollup:**
  - If the group status is Critical, all members are in the critical state.
  - If the group’s status is anything other than Critical, at least one member is in the displayed state. No members are better than the displayed status. It is also possible that all members are in this state.

- **For Worst status rollup:**
  - If the group status is Up, all members are in the Up state.
  - If the group’s status is anything other than Up, at least one member is in the displayed state. No members are worse than the displayed status. It is also possible that all members are in this state.

- **For Mixed status rollup:**
  - If all members are in the same status, that status will be the group status.
  - If the group status is warning, either the group contain items of differing status (most common), or all members have the status of Warning (uncommon).

While this may seem complicated, the logic for choosing which type of rollup is fairly straight forward.

- **For a group where every direct member (not member of a subgroup) is critical to be in the Up state,** choose Worst status rollup. This will ensure that if any member has an issue, you will see that reflected in the group status and any alerts or reports created for that group.

- **For groups with redundant member resources,** such as a dual attached WAN, choose Mixed or Worst status rollup, depending on the criticality of a worst-case, single-member failure.
For groups with a high level of redundancy throughout all direct members, choose best status.

The reason why we specify direct members and subgroup members is to allow for the group status rollup to be an additive rollup, from the lowest level subgroup to the top-level group. Take the following datacenter (DC1) for example.

Let's create one top-level group called DC1 and then create member subgroups for all the like items. The choices are many. We could create groups for each redundant server pair or a group for all redundant server pairs. The non-redundant servers could exist as individual objects or as one or more groups. To make a plan on how to arrange these, we will first consider our goal; to manage the data center network. This means that at this time we are not concerned with the processes that are enabled by the network, just that the network is available and performing well. In examining the naming conventions for the data center switches we find that they have well planned and consistent device names as follows:

- All core switches are named DC1-core-xx, where xx is the core switch number.
- All service switches are named DC1-service-xx, where xx is the service switch number.
- All distribution switches are named DC1-dist-xx, where xx is the distribution switch number.

With this in mind we create the three dynamic service groups for the above items.
Dynamic service group DC1 Core where a query for “system name contains DC1-core”. Likewise groups are made for the service switches and distribution switches. Now we'll look at the servers. In speaking with the server owners, they state that they don't care if a server is redundant or not. If a server is having a problem, they want to be able to see that from all levels. With this information we set the servers subgroups to all show worst member status as group status. We find we can add the redundant servers using a dynamic query, but unfortunately we are unable to identify any common and unique qualifiers for the non-redundant servers, so these servers will be added statically, as individual members of a DC1- non-redundant-servers subgroup of DC1. The only items left are the switch ports and links to the corporate network. These have good consistent port descriptions which allow us to create port and port type groups. Seeing this, we create dynamic groups for the corporate network ports, the core ports, and the distribution ports.

By looking at all the network equipment, heavy use of connection redundancy, we take the same path as the server teams and set the subgroups status in each case to reflect the worst member status. Then we also set the status of the DC1 group to the status of the worst member. Here we have taken the most conservative approach to managing the group status. When any object in any of these groups fails or slows enough to trigger a threshold, we will see that status reflected as the status of DC1. But is this a wise idea? While we do want to quickly find and identify the failed element in DC1, having the group status set to the worst status will probably indicate that DC1 status in Up (green) or the DC1 status is Down (red), Warning (yellow) when in all three of these possible cases DC1 as a whole is perfectly operational. A better choice would be to keep the subgroups as Worst status and set the status of DC1 to Mixed. In so doing, DC1 will be green when every element of the group is Up and will have a warning status if there are elements with a status lower that Up. Perhaps the worst choice would be to set the DC1 group status to Best. If we did this, DC1 would always have an Up status, even if every member but only one has failed.

Here, we have created one top-level group and ten subgroups, all at the second level. You can choose to embed subgroups as far as you want into other subgroups. There is no hard limit, but as you embed groups deeper, the logic becomes more complex. Objects can also be members of multiple groups. Implementing reports indicating group membership and careful examination of existing and planned groups is recommended.

A couple rules of thumb should be considered when creating subgroups.

1. Determine if you can accomplish the same goal without using subgroups.

2. Keep the subgrouping as flat and as simple as possible. The more subgrouping levels, the more difficult it is to understand the logic flow from one level to higher levels. Depending on the complexity of the subgroups, the logic can increase as much as \( n^2 \), where \( n \) is the number of group layers. The dependencies logic will also become complicated.

Perhaps after setting up this grouping and rolling out the status to maps and user views, you get a complaint from the inventory management department. Their complaint is that it is hard to see in the current grouping, if there is a problem directly affecting their data processing done in the datacenter. Because they are such a small portion of the data center, they must investigate what caused DC-1’s status to change to see if any of their critical devices are in trouble. This is time consuming and causes what they are calling false positives.
This department uses two non-redundant application servers, the clustered database, an IP path to the input web portal offsite and an IP path to a business partner connection. They don't really care to see that a redundant link or redundant equipment is down. They just want to know if the inventory management system is working or not. With this in mind, here is what we create. First, we create a DC-1 Inventory-Mgmt group. Then, we add the same groups for the entire redundant network infrastructure as we did in the DC-1 group, but we set the status of each to *Best*. This is because they are only interested in knowing the datacenter network works for what they need. With the high level of redundancy, chances are, the best rollup status for these items will always be *Up*. We don’t need to add the redundant servers, as they don’t use those. Then, we add the DB cluster and individual application servers as individual static members of DC-1 Inventory-Mgmt group. Now, we set the DC-1 Inventory-Mgmt group rollup status. Because the servers are non-redundant, we need to show that there is a problem with those or with the database. Therefore, we set the DC-1 Inventory-Mgmt top-level rollup to *Worst*. Now if any single, non-redundant portion fails or if there are any complete failures across a redundant portion of the network, the top-level group will indicate a failure in Inventory Management within the datacenter. But, what about the partner connection and web interface? If we could add the business partner interface management and portal testing as part of our group, this would give a much more complete picture of the abilities of the network to support the Inventory Management business task.

The group function built into Orion core allows you to add objects from Orion modules.
What we do now is create an ICMP echo IP SLA operation in Orion IP SLA Manager from a point in the datacenter to the internal business partner connection port. Intra-data center IP traffic normally has round trip times measured in microseconds, so it doesn’t really matter where in the database we place the operation. After creating the operation, we add it to the DC-1 Inventory-Mgmt group. Next, using the Orion Application Monitor, we add a user experience test for logging into the inventory manager web interface. This test is then added as an object into the DC-1 Inventory-Mgmt group. We may also add statistics on the application servers’ volumes to the DC-1 Inventory-Mgmt group. Here is the final grouping.

- DC-1 Inventory-Mgmt Group. Status Rollup = Worst member status
  - App server #1 as member
  - App server #2 as member
  - All server volumes (#1 and #2) as members
  - All direct connections to server as individual members
  - IP SLA ICMP echo operation as member
  - APM web test as member
  - Core switch group as member. Status rollup = Best member status
  - Core switch ports group as member. Status rollup = Best member status
  - Service switch group as member. Status rollup = Best member status
  - Service switch ports group as member. Status rollup = Best member status
  - Distribution switch ports group as member. Status rollup = Best member status
  - Distribution switch group as member. Status rollup = Best member status

You would probably further group the switch ports by function (inter core connection, core to service, service to distribution, et cetera), but I have not added those as they are not necessary to show the functions of groups and subgroups. So, we have created a datacenter network management group and an inventory management business process management group. Each has its own goal and functions to meet the needs of each party. There are objects that are members on multiple groups, groups with different rollup status as well as static and dynamic members of the groups. Using groups is a powerful feature for organizing objects and adding logic to the relationships between objects. It also enables another powerful feature of Orion – dependencies.

**Dependencies**

Orion allows for two types of dependencies, implicit dependencies and explicit dependencies. Implicit dependencies are part of the Orion code and are implemented automatically. These dependencies handle cases in which objects are always dependent on high level (parent) objects. Volumes and interfaces are implicitly child objects of the node they belong to. For Application Performance Monitor, applications are implicitly children of a node status as well. In IP SLA Manager, IP SLA operations are implicit children to the status of the node at both ends of an operation.

Explicit dependencies are dependencies you define. They operate by checking the status of a parent object you have defined. If the parent object status is *Unreachable or Down*, the child is set to *Unreachable*.

*Unreachable* is a new status that is not propagated by Orion alerting by default, allowing for a suppression of false positive alerts. Whenever an objects parent object has failed, or is part of a larger failure, the status assignment of *Unreachable* to all child objects prevents a cascade of child object alerts. This is most commonly used in the case where a portion of the network is separated from the location housing the Orion server by a WAN connection.
Let’s consider remote site A. This site is connected to the headquarters, where the Orion server is installed, by two WAN connections. In site A there are several switches, servers and network users. Site A is completely dependent on the two WAN connections back to headquarters for all operations. To create a dependency which reflects this we will first create a group for the two WAN ports at the headquarters and call it Site A WAN. Then we will create a group called Site A and add all managed objects completely contained in Site A. The end of the WAN connections will be made into a group called Site A WAN. That group will only contain the two WAN interfaces at the headquarters that are connected to Site A.
Both groups will use Mixed status rollup. We go on to create a dependency for the Site A group as the child of the Site A WAN group. And then, we create an alert for Site A ensuring we are alerted when the site A group status in not equal to *Up* and indicate in the alert action the status of the offending managed object. This will alert when site A has an issue with any object of site A and tell us what is causing it. If the WAN connections fail to site A, the interfaces on the headquarters side will go down and we will only receive an alert that the WAN group has failed. We can insert an alert action noting that site A is now unreachable.

In an alternate solution which yields more granularity, we would group like items in Site A and create dependencies which flow from the WAN links down into Site one layer at a time. Here is what you would create to accomplish this.

- Group “Site A WAN”, containing WAN links to site A
- Group Site A Routers, containing the routers
- Group Site A Switches, containing the Switches
- Group Site A Servers, containing the Servers

Then we would create the following dependencies

<table>
<thead>
<tr>
<th>Parent</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site A WAN</td>
<td>Site A Routers</td>
</tr>
<tr>
<td>Site A Routers</td>
<td>Site A Switches</td>
</tr>
<tr>
<td>Site A Switches</td>
<td>Site A Servers</td>
</tr>
</tbody>
</table>

Keep in mind that you can assign multiple parents by membership in multiple groups which may cause difficulties in troubleshooting dependency problems. Because of the flexibility built into these features, you may create very complex and interdependent groups and dependencies. Whenever you are creating groups and dependencies, the simplest method will be the best. Most groupings designs can be achieved with only a single layer of sub grouping as shown on pages 6 and 7.

**Alerting and Reporting**

Both the Orion Advanced Alert Manager and Report Writer have been updated to include grouping capabilities. For information on groups and alerts please see *Understanding Orion Advanced Alerts*.

For information on groups and reports please see *Understanding Orion Report Writer*. 