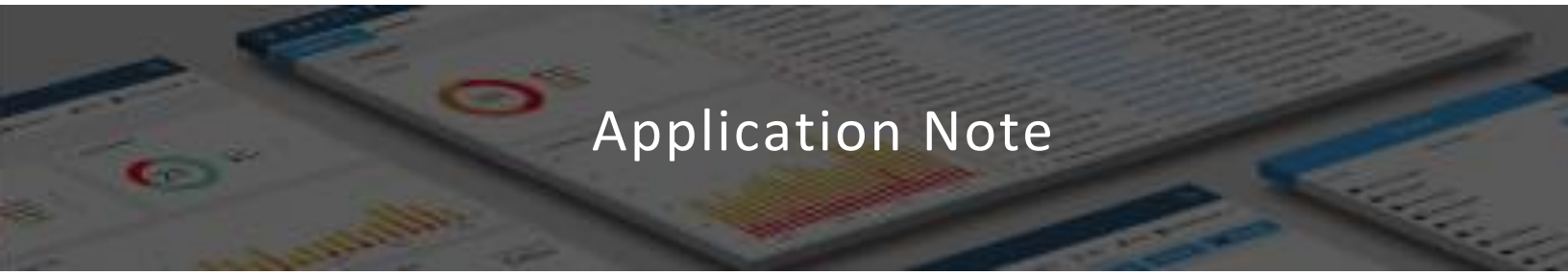
ENTUITY
NETWORK ANALYTICS



Application Note

Creating a report that shows counts of ports of various categories using the Report Builder

15th April 2019

John Diamond

Table of Contents

1 Overview

This Application Note describes an approach to creating a report using the Report Builder that will iterate through multiple devices and for each one will identify the device by name and show the counts of ports in each of the following four categories:

- Ports with descriptions starting with “Gi” that are operationally down
- Ports with descriptions starting with “Gi” that are operationally up
- Ports with descriptions starting with “Fa” that are operationally down
- Ports with descriptions starting with “Fa” that are operationally up

2 Challenges to be overcome

i) There are no standard device attributes that contain the required port counts
ii) Some types of device such as Ping-only and Custom don’t have associations to ports

3 Developing and testing StormWorks Statement Language syntax

This report will need to be created using the Report Builder. The columns containing the port counts will need to have custom attributes defined that implement the logic necessary to compute the individual counts. This logic is specified using StormWorks Statement Language which is the syntax used by the Data Management Kernel to specify the data models that every monitored component is represented by. Details of the Statement Language syntax are described in a separate document.

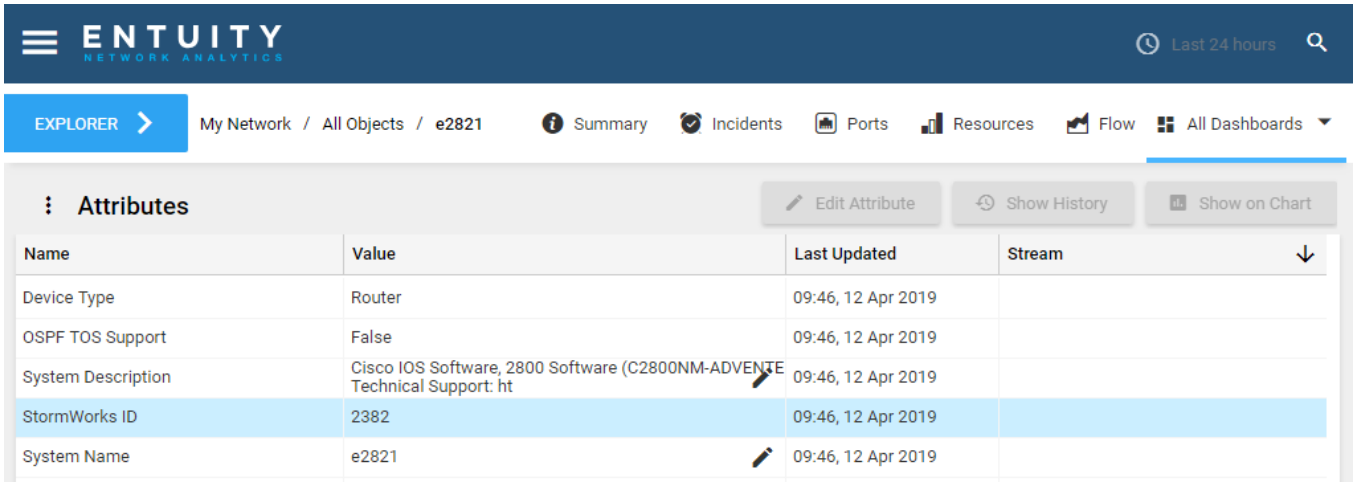
You’ll need a way to test the syntax as you create it. This can be done using a Telnet client such as the one provided with the Windows package. Under Windows, you’ll need to install the Telnet client if it isn’t already installed as it doesn’t get installed in the default Windows configuration. When using the Windows Telnet client, you should launch it from a Command Prompt as follows:

telnet <EntuityServer> 5465

Note that although this is using Telnet protocol it is connecting to TCP port 5465 which allows direct access to the Data Management Kernel. This is referred to as the “debug port”. The client window should initially be blank, and you’ll need to type an ‘x’ to display the login prompt. When you enter a valid user name and password you’ll receive a “$” prompt. Note that user name and password are for an Entuity account not the host operating system. Any command entered at this prompt will be executed by the Data Management Kernel. If you use a different Telnet client such as the one available under Linux you should enter a ‘t’ rather than an ‘x’ to gain proper access.

When computing a port count for a device you’ll need to be operating in the context of a device. When testing/debugging statement language syntax, the current context can be set by typing “id=<StormWorks object ID>”. Each component/object known to the system has a unique StormWorks object Id (swId) within that server.

One way to obtain the swId for a device is to navigate to a device using the Explorer in the web UI and select the Attributes dashboard. The StormWorks ID is displayed as one of the attributes:



An alternative approach to obtaining the swId for a device called “e2821” would be to issue the following command via the debug port:

```
get_device(“e2821”)
```

The result will look like this:

```
Cacheable(0x8002=CT_OBJECTINSTANCE,2382)
```

The swId, in this case, would be 2382.

4 Building the command to count ports via the debug port

Fully managed devices have a StormWorks Object that represents the device itself. This device Object contains no information about the ports on the device. Each port is modelled by a port Object. There is an Association between each fully managed device Object and its port Objects. In order to evaluate how many of the ports meet the various criteria required for this report, each one will need to be checked to see if its description has the appropriate “Gi” or “Fa” at the beginning of the string and that the operational state is either UP or DOWN. It’s worth noting that the operational state can potentially be in one of a number of different states, so I’ll be looking for UP and not UP.

4.1 Obtaining a list of port objects for a device

If the current context is a fully managed device the list of port objects associated with that device can be obtained using “ref.ports”. Having set the context to be a fully managed device using the “id=<swId>” syntax you can test this by entering “simple;ref.ports”. The “simple;” is used to indicate that the rest of the line is to be interpreted using the “simple” language which is not the default language used by the Data Management Kernel. The language used if you don’t enter “simple;” is referred to as the “Native” language. Although StormWorks functions can be called using the Native language and the context can be set using “id=<swId>” you can’t do

much else so almost every operation will start with “simple;”. If you’ve used the Windows Telnet client to connect to the debug port, you can take advantage of the TAB key to act as a command line completion accelerator. If you type TAB at the beginning of a line it will automatically enter “simple;” for you. If you enter enough characters of a function name to uniquely identify it followed by the TAB key it will be automatically completed. Here’s an example of how the list of port Objects were displayed at the debug port:

```
$get_device("e2821")

Cacheable(0x8002=CT_OBJECTINSTANCE,2382)

$id=2382

$simple;ref.ports

(Cacheable(0x8002=CT_OBJECTINSTANCE,2404), Cacheable(0x8002=CT_OBJECTINSTANCE,2405),
Cacheable(0x8002=CT_OBJECTINSTANCE,2406), Cacheable(0x8002=CT_OBJECTINSTANCE,2407),
Cacheable(0x8002=CT_OBJECTINSTANCE,2408), Cacheable(0x8002=CT_OBJECTINSTANCE,2409),
Cacheable(0x8002=CT_OBJECTINSTANCE,2410), Cacheable(0x8002=CT_OBJECTINSTANCE,2411),
Cacheable(0x8002=CT_OBJECTINSTANCE,2412), Cacheable(0x8002=CT_OBJECTINSTANCE,2413),
Cacheable(0x8002=CT_OBJECTINSTANCE,2415), Cacheable(0x8002=CT_OBJECTINSTANCE,2416))
```

The list of port Objects is enclosed within parentheses indicating that it is a list. The number of items in the list can be computed by passing it to the “count” function as a parameter:

```
$simple;count(ref.ports)

12
```

There is an alternative way to obtain the list of port Objects associated with a device:

```
$simple;get_associated_objects("ports")

(Cacheable(0x8002=CT_OBJECTINSTANCE,2404), Cacheable(0x8002=CT_OBJECTINSTANCE,2405),
Cacheable(0x8002=CT_OBJECTINSTANCE,2406), Cacheable(0x8002=CT_OBJECTINSTANCE,2407),
Cacheable(0x8002=CT_OBJECTINSTANCE,2408), Cacheable(0x8002=CT_OBJECTINSTANCE,2409),
Cacheable(0x8002=CT_OBJECTINSTANCE,2410), Cacheable(0x8002=CT_OBJECTINSTANCE,2411),
Cacheable(0x8002=CT_OBJECTINSTANCE,2412), Cacheable(0x8002=CT_OBJECTINSTANCE,2413),
Cacheable(0x8002=CT_OBJECTINSTANCE,2415), Cacheable(0x8002=CT_OBJECTINSTANCE,2416))
```

Although, as first glance, it might appear to be directly equivalent to “ref.ports”, the difference becomes apparent if, instead of a fully managed device, the current context is a Ping-only device which doesn’t support the concept of associated ports. Executing “ref.ports” will raise a runtime error whereas get_associated_objects("ports") will return an empty list. If you were to create a report based on Fully Managed Devices, then either approach would work acceptably. If the report were based on all types of devices including those that don’t have associated ports the result would likely be that error messages would appear for those port-less devices. For this reason get_associated_objects("ports") will be used.

4.2 Iterating over the list of ports so that each can be separately evaluated

In order that a count of the “Gi” ports that are operationally DOWN (not UP) can be performed, the complete list of ports needs to be reduced to only representing those that match that criteria. This means that we need to iterate over the port Objects in the list performing the relevant tests on each. This can be achieved using the “foreach” function. The “foreach” function takes either two or three parameters and returns another list. The first parameter is a list and the second is what should be returned for each item in the list. If a third parameter is supplied, it is treated as a Boolean (returns true/false). The resulting list is only populated with those items for which the third parameter returns “true”. This means that if the logic to identify whether a ports description begins with “Gi” and the operational state is not UP is placed in the third parameter it will result in the foreach function returning a list where each element in the list represent a port matching that criteria.

When the second and third parameter are evaluated at runtime, a special object called “this” represents the item in the list that is being evaluated. This means that for each of the items in the supplied list the second parameter will be evaluated, and the results placed in the list that it returned upon completion. Where “this” represents an Object, as in this case, it must be typecast to the appropriate Object type. The port objects are of type “portEx” so the required syntax would be “portEx(this)”. The description of the port, as obtained from the MIB-2 ifDescr OID, is held in an Attribute of the portEx Object called “portDescr”. This can be obtained using “portEx(this).portDescr”. To illustrate this mechanism, you can enter the following to return a list of port descriptions for the device that is the current context:

```
$simple;foreach(get_associated_objects("ports"),portEx(this).portDescr)
```

(GigabitEthernet0/0, GigabitEthernet0/1, Serial0/0/0, Serial0/0/1, Serial0/1/0, Serial0/1/1, Null0, Loopback0, Loopback1, Loopback999, GigabitEthernet0/1-mpls layer, Serial0/1/0-mpls layer)

4.3 Matching on part of a string

There is a requirement to test whether each port description starts with “Gi”. This can be achieved using a regular expression test. The “regex” function is available for this purpose and takes two parameters. The first parameter is the regular expression pattern and the second is the string to be tested. The “regex” function returns a structure (very similar to a list) containing three elements. This can be illustrated as follows:

```
$simple;regex("^Gi","GigabitEthernet0/0")
```

{Gi, 0, 2}

```
$simple;regex("^Gi","Serial0/0/0")
```

{, -1, 0}

The three elements of the returned structure represent the extracted string, the position in the string where the match was found and the number of returned characters. If there is no match, then the second element is -1. In this case the pattern “^Gi” only matches “Gi” if it is at the start of the string so the second element will either be 0 for a match or -1 where there is no match.

The second element of the returned structure can be extracted by appending “[1]” to the function call:

```
$simple;regex("^Gi","GigabitEthernet0/0")[1]

0
```

A test for equality is performed using “==”. A list of port descriptions for only those ports whose descriptions begin with “Gi” can therefore be achieve using:

```
$simple;foreach(get_associated_objects("ports"),portEx(this).portDescr,regex("^Gi",portEx(this).portDescr)[1]==0)

(GigabitEthernet0/0, GigabitEthernet0/1, GigabitEthernet0/1-mpls layer)
```

The operational state of a port is obtained from the portOperationalStatus Attribute. It is returned as an integer where 1=UP, 2=DOWN and various additional values have other non-UP meanings. A test for inequality can be made using “!=”. The iteration syntax can be extended to not only check for “Gi” but also test that the port is not up as follows:

```
$simple;foreach(get_associated_objects("ports"),portEx(this).portDescr,regex("^Gi",portEx(this).portDescr)[1]==0 && portEx(this).portOperationalStatus!=1)

()
```

In this case, all three of the Gi ports are operationally UP so the returned list is empty. The “&&” is the logical AND operator which means both the string match and the operational status checks must have evaluated as “true” in order for the entire expression to be considered “true”.

As the sole purpose of creating a list, in this example, is to be able to count the elements within it, the contents of the elements in the list can be safely simplified to just a number such as the swId of the port which can be obtained using “id”. I’ve now changed the context to a device which has four Gi ports that are down. When wrapped in a “count” function, it would look like this”:

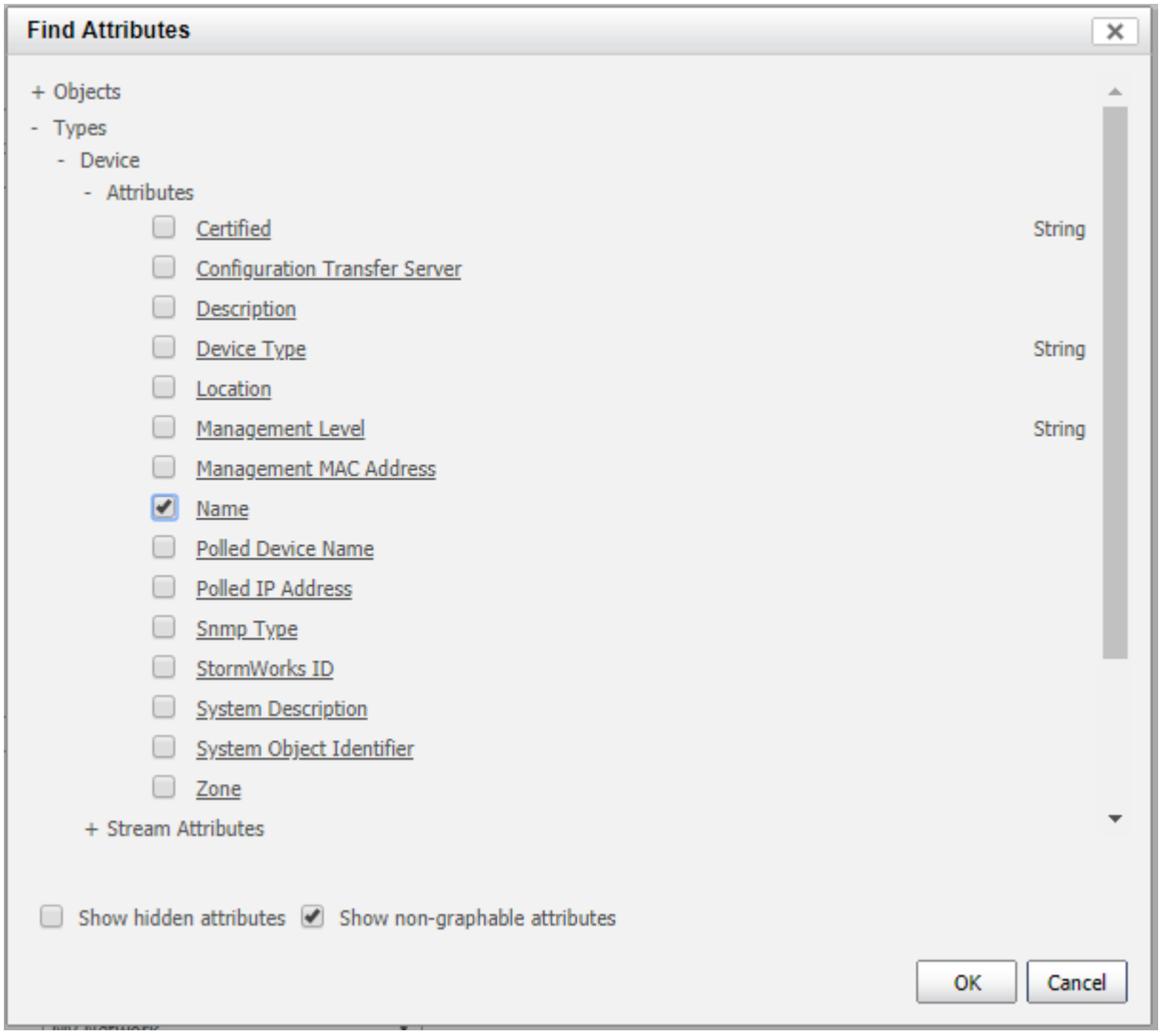
```
$simple;count(foreach(get_associated_objects("ports"),id,regex("^Gi",portEx(this).portDescr)[1]==0 && portEx(this).portOperationalStatus!=1))

4
```

The equivalent syntax to count the Gi ports that are UP would be:

```
$simple;count(foreach(get_associated_objects("ports"),id,regex("^Gi",portEx(this).portDescr)[1]==0 && portEx(this).portOperationalStatus==1))
```

5 Building the report

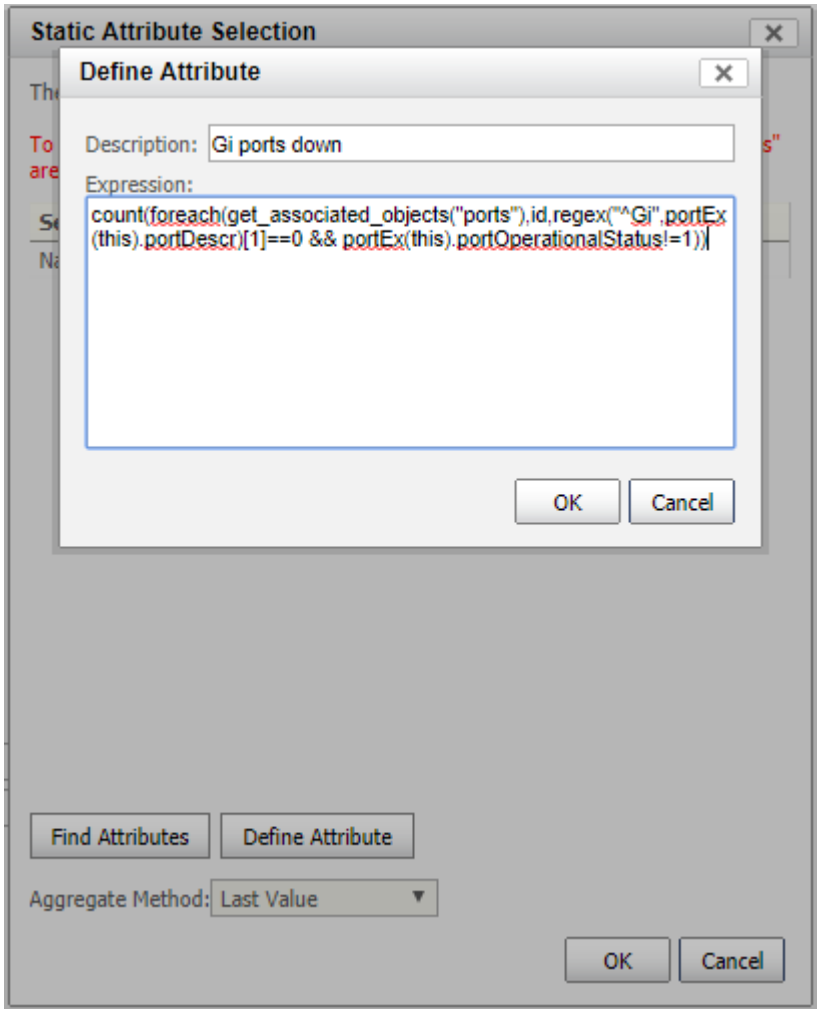
Launch the Report Builder using ***Main Menu -> Reports -> Report Builder*** and select the Inventory table report template. Select the "Ignore selection, and use objects from view" option. Leave the default object type as "All Devices". The various columns of the report table can be defined and will represent the device name followed by the four different port counts. To add the device name column, click the "Add Column" button then the None link in the "Value" column to display the Static Attribute Selection dialog. Click the "Find Attributes" button then click the "+" to the left of "Types", "Device" and "Attributes". Check the box to the left of "Name" and click the "OK" button:



Select "Name" in the Selections column and the "OK" button.

To create a column for the count of Gi ports that are down, click the Add Column. For this attribute we won't be able to find it in the list of standard attributes so it will need to be manually defined using the previously developed Statement Language syntax. Check the Show Advanced Options box in the top right of the page. Now

when the None link is clicked an extra button labelled “Define Attribute” appears in the dialog. Click this button and enter a suitable description for the Attribute such as “Gi ports down”. The previously developed syntax for counting the number of Gi ports that are down should be placed into the Expression field but do not include the “simple;” on the front. Click “OK” on this dialog and “OK” on the previous dialog.



The same approach can be used to create the other columns where the Expression syntax will vary slightly to test for either “Gi” or “Fa” and the test for the port status will be either “!=1” or “==1”.

Finally, the report can be given a name and, optionally, a title before being published.

It should be noted that, because the list of associated ports for each device is being obtained using get_associated_objects("ports"), those devices without ports will simply show the port counts as zero.

6 Finished report

Entuity Report

Inventory of Gi and Fa ports with status

Over the period 14:00 on Thu Apr 11 2019 - 14:00 on Fri Apr 12 2019

No prime time is set for this report

Printed on: 12 Apr 2019 14:54:27 EDT

Name	Gi ports down	Gi ports up	Fa ports down	Fa ports up
bottom2960	1	0	19	5
bottom3550	0	2	22	2
new2610	0	0	0	2
r2610	0	0	0	1
stack3750	4	0	31	17
subzero	0	0	0	0
top2960	0	2	12	12
10.44.1.43	0	0	3	0
BOSTON-ROUTER	0	0	0	0
CHICAGO-ROUTER	0	0	0	0
CHICAGO-SWITCH	0	0	10	2
CHICAGO-SWITCH2	2	0	23	1
SANFRAN-ROUTER	0	0	0	0
SANFRAN-SWITCH	0	0	10	2
ramsey-ap	0	0	0	1