IPSWITCH

Monitoring High Capacity Counters with WhatsUp Gold v11

IPSWITCH WhatsUpGold

Monitoring High Capacity Counters using WhatsUp Gold v11

In this Document

Introduction

This paper demonstrates how to monitor high capacity (64-bit) counters using active monitors in Ipswitch WhatsUp Gold v11.

While this paper describes the process for monitoring traffic counters for gigabit ethernet interfaces, the principles discussed can be applied to monitor any high capacity counters available via SNMP or WMI.

SNMP background

High capacity counters were introduced in SNMPv2 to address the limitations of 32-bit counters for devices that track so much data that the counter reaches its maximum value and wraps back to zero.

This effect is evident in network interface utilization, where faster interface speeds cause 32-bit counters to wrap so quickly that it is difficult to poll the counter frequently enough to not miss a counter wrap.

Interface speed	**Time to wrap**
10Mbs	~ 57 minutes
100Mbs	~ 5.7 minutes
Gigabit (1Gbs)	~ 34 seconds

High capacity counters alleviate this by substantially increasing the size of the counter: To wrap a high capacity counter every 5 years, for example, would require an interface operating at speeds of 1Tbs (1000Gbs).

Your installation of WhatsUp Gold already includes the MIB file required to monitor high capacity utilization counters for network interfaces that support them. You can view the MIB file, name *IF-MIB.txt* in the *Data\Mibs* subdirectory of the WhatsUp Gold installation directory (typically *C:\Program Files\Ipswitch\WhatsUp\Data\Mibs*).

This MIB contains three counters necessary for monitoring network utilization using high capacity counters, **ifHCinOctets** and **ifHCoutOctets**, which contain information about the data passing through the interface, and **ifHighSpeed**, which is used to calculate the capacity of the interface. With these three values, the total interface utilization can be calculated and expressed as a percentage of the interface's total capacity.

Note: To read high capacity counters, an SNMPv2 or higher credential must be provided. SNMPv1 does not support high capacity counters.

ifHCinOctets

Object ID	1.3.6.1.2.1.31.1.1.1.6
Label	iso.org.dod.internet.mgmt.mib-2.ifMIB.ifMIBObjects.ifXTable.ifXEntry.ifHCInOctets
Type	64bit Counter
Access	Read Only
Description	The total number of octets received on the interface, including framing characters. This object is a 64-bit version of ifInOctets. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounterDiscontinuityTime.
MIB Module	IF-MIB

ifHCoutOctets

Object ID	1.3.6.1.2.1.31.1.1.1.10
Label	iso.org.dod.internet.mgmt.mib-2.ifMIB.ifMIBObjects.ifXTable.ifXEntry.ifHCOutOctets
Type	64bit Counter
Access	Read Only
Description	The total number of octets transmitted out of the interface, including framing characters. This object is a 64-bit version of ifOutOctets. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounterDiscontinuityTime.
MIB Module	IF-MIB

ifHighSpeed

Object ID	1.3.6.1.2.1.31.1.1.1.15
Label	iso.org.dod.internet.mgmt.mib-2.ifMIB.ifMIBObjects.ifXTable.ifXEntry.ifHighSpeed
Type	Gauge
Access	Read only

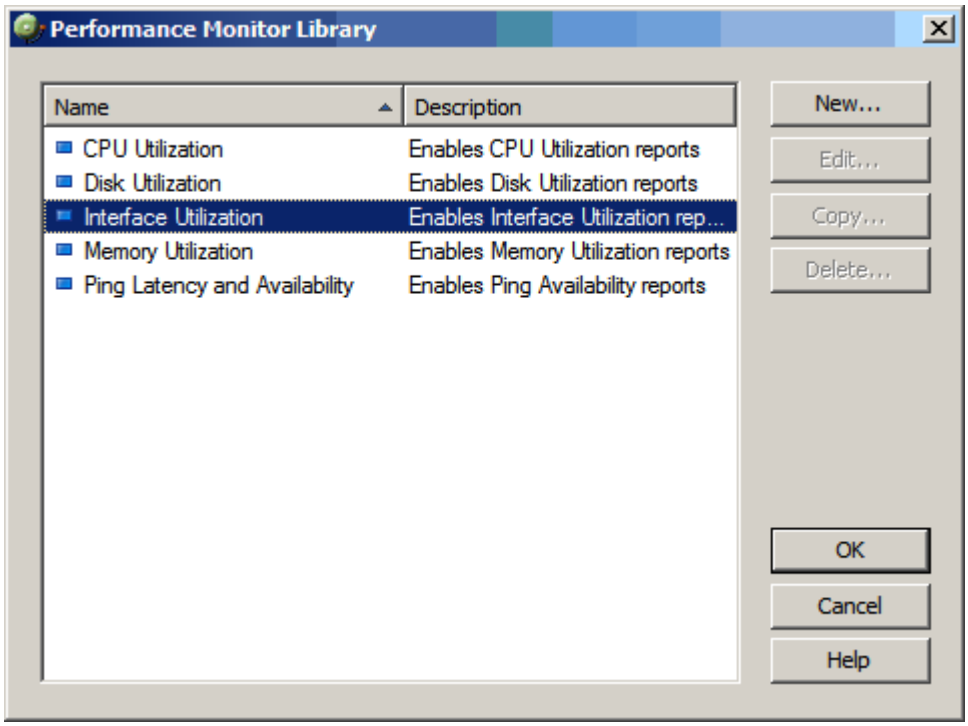
Description An estimate of the interface's current bandwidth in units of 1,000,000 bits per second. If this object reports a value of `n' then the speed of the interface is somewhere in the range of `n-500,000' to `n+499,999'. For interfaces which do not vary in bandwidth or for those where no accurate estimation can be made, this object should contain the nominal bandwidth. For a sub-layer which has no concept of bandwidth, this object should be zero.

MIB Module IF-MIB

These three OIDs, plus the instance numbers (which indicate the interface from which the counter should be read), provide the information necessary to monitor network utilization on an interface that support high capacity counters.

WhatsUp Gold v11 and standard counters

WhatsUp Gold v11 provides native support for monitoring 32-bit interface utilization counters using performance monitors. By default, the performance monitor library includes the interface utilization monitor, which can be applied to any device to monitor interface utilization using standard 32-bit counters.



Monitoring high capacity (64-bit) counters using WhatsUp Gold v11 requires customizations to the base product, which are outlined in the following sections.

Monitoring interface utilization using high capacity counters

Since SNMP breaks interface utilization into inbound (represented by `ifHCinOctects`) and outbound (represented by `ifHCoutOctects`), monitoring total network interface utilization requires monitoring both of those values.

To monitor interface utilization as a percentage of total capacity, the capacity of the interface--its speed--must also be known.

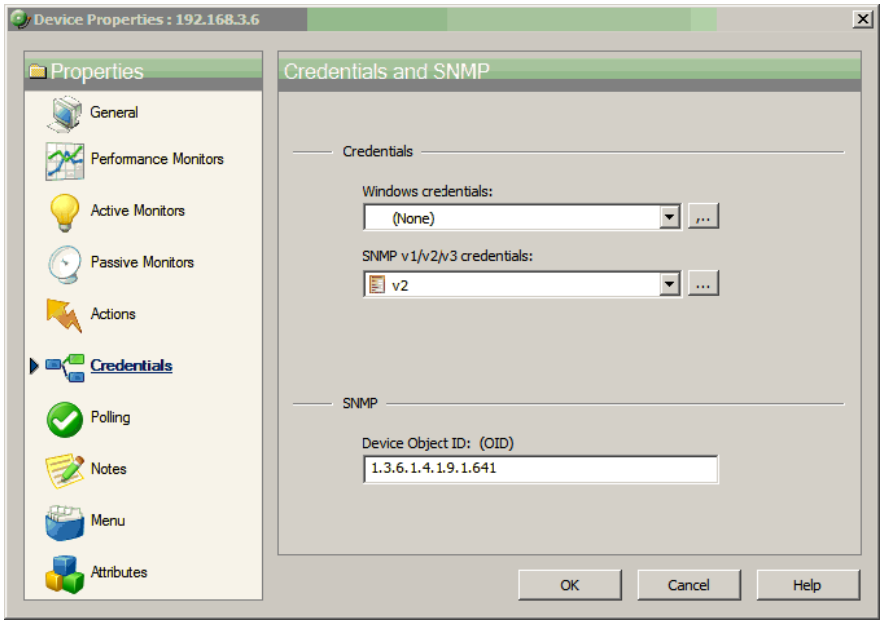
Using performance monitors to graph interface utilization over time

You can use a performance monitor to watch interface utilization over time. The performance monitor creates a report that graphs the results of the monitor, allowing you to spot trends and patterns.

These steps will guide you through the process of creating a performance monitor to track interface utilization over time.

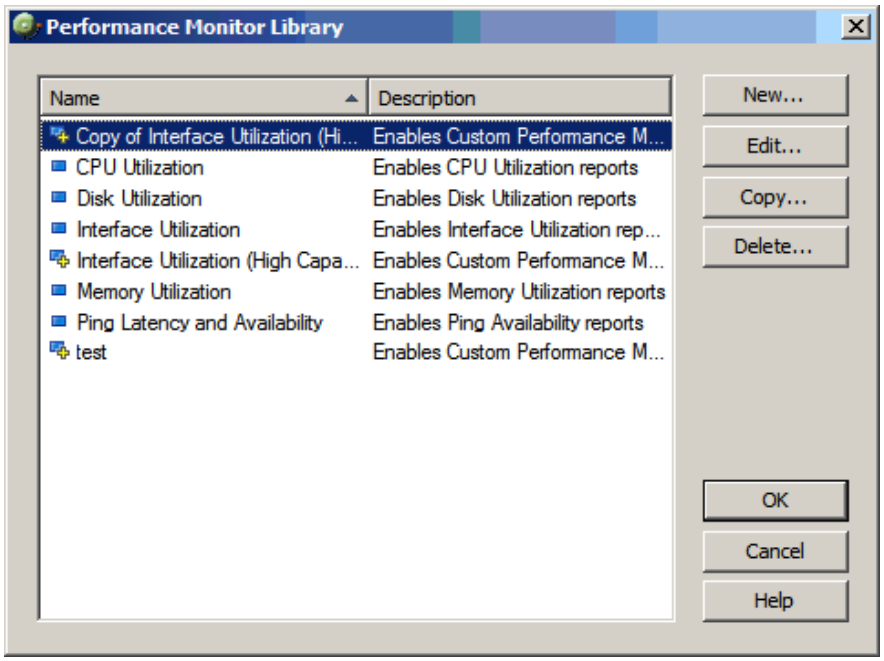
Step 1: Verify the device is using SNMPv2 or SNMPv3 credentials

1. Open the device properties for a device that supports high capacity counters for interface utilization.
2. Select **Credentials**. The Credentials and SNMP dialog opens.
3. In **SNMP v1/v2/v3 credentials**, verify that the device is using an SNMPv2 or SNMPv3 credential. If an SNMPv2 or SNMPv3 credential does not exist, you can create one by clicking the **Browse (...)** button next to the field.



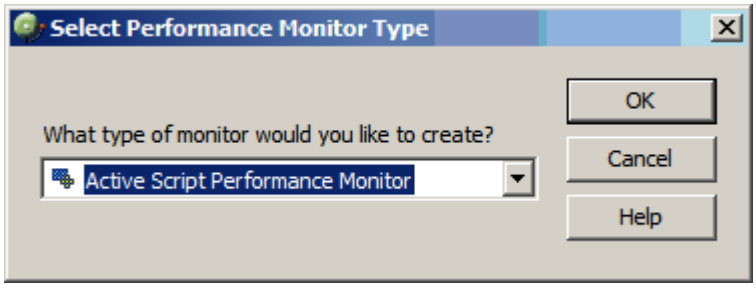
Step 2: Create a new script in the Performance Monitor Library

1 From the main menu, select **Configure > Performance Monitor Library**. The Performance Monitor Library dialog opens.



2 Click **New**. The Select Performance Monitor Type dialog opens.

3 Select **Active Script Performance Monitor**, then click **OK**. The Add Active Script Performance Monitor dialog opens.



4 Give the monitor a **Name** and **Description**, and select **JScript** as the **Script type**.

5 To access the data contained in the high capacity counters, create a **Reference Variable** for each.

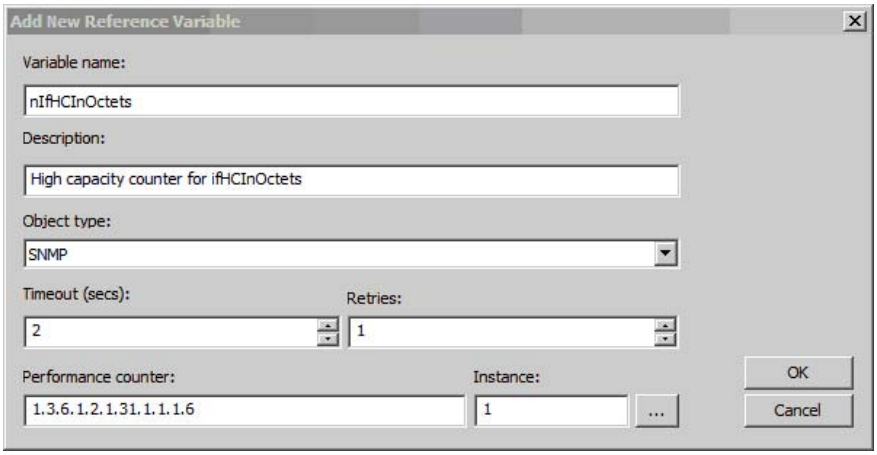
   a) Click **Add**. The Add New Reference Variable dialog appears.

   b) Enter a **Name** and **Description** for the variable.

   c) In **Performance Counter**, enter the OID for one of the high capacity counters you want to monitor.

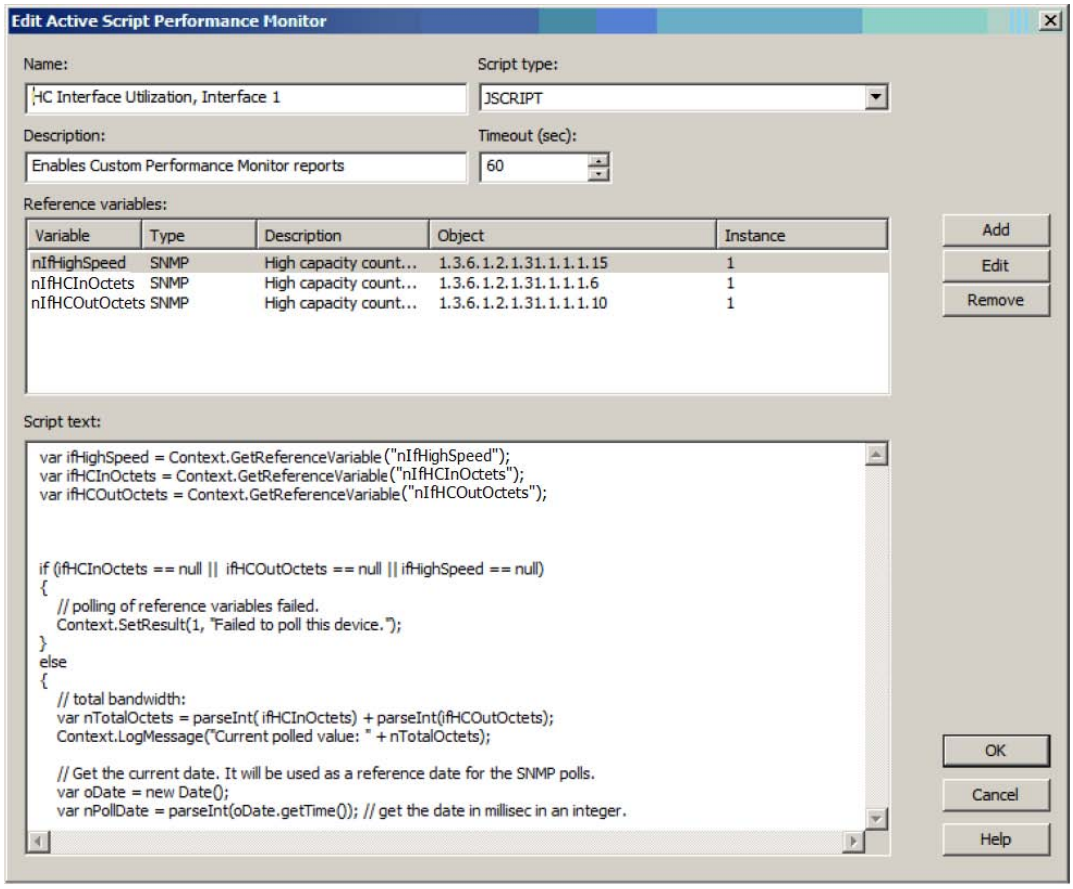
   d) In **Instance**, enter the instance number of the interface that you want to monitor.

e) Click **OK** to save the reference variable. The Add Reference Variable dialog closes and the Add Active Script Performance Monitor dialog reappears.



f) Repeat steps 1-5 for each of the high performance counters (`nIfHCInOctets`, `nIfHCOutOctets`, and `nIfHighSpeed`).

6 With all three reference variables in place, enter into **Script body** the script example included at the end of this document.



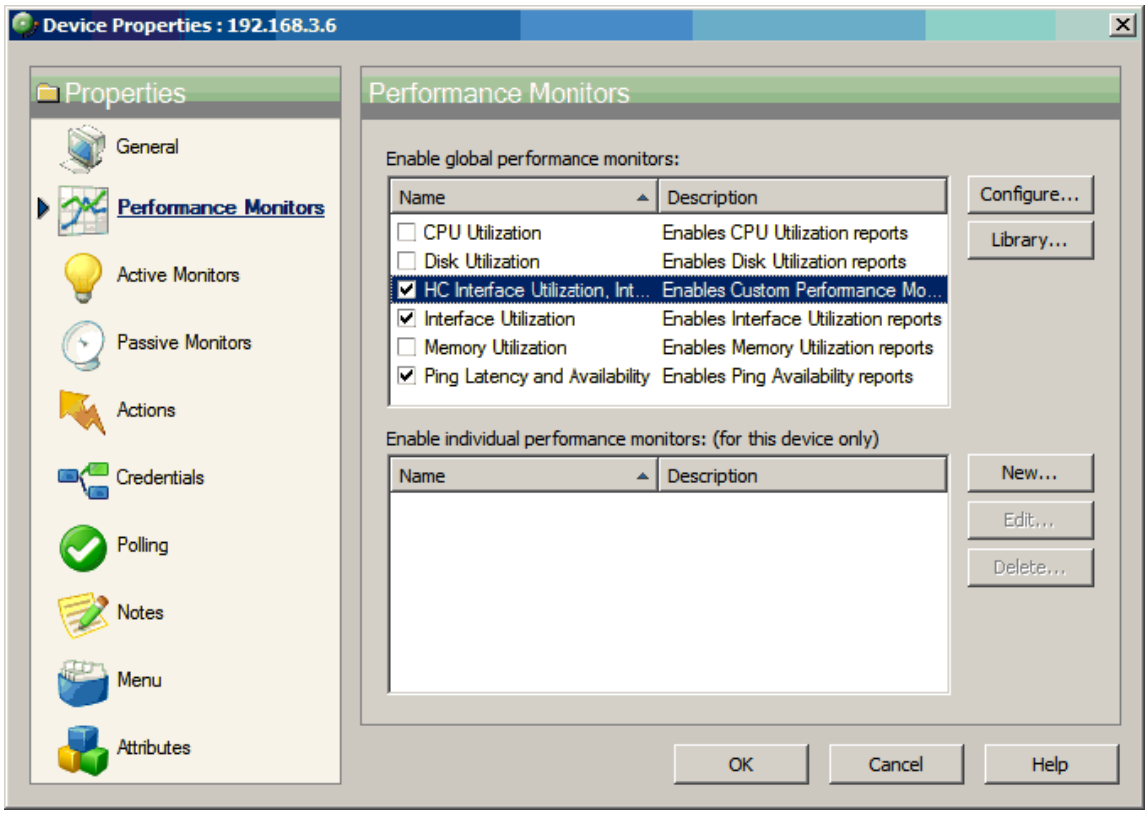
7 Click **OK** to save the Active Script Performance Monitor.

Step 3: Apply Active Script Performance Monitor to device

1 Open the device properties for the device you want to monitor using the new monitor.

2 Select **Performance Monitors**. The Performance Monitors dialog appears.

3 Under **Enable global performance monitors**, select the new monitor.



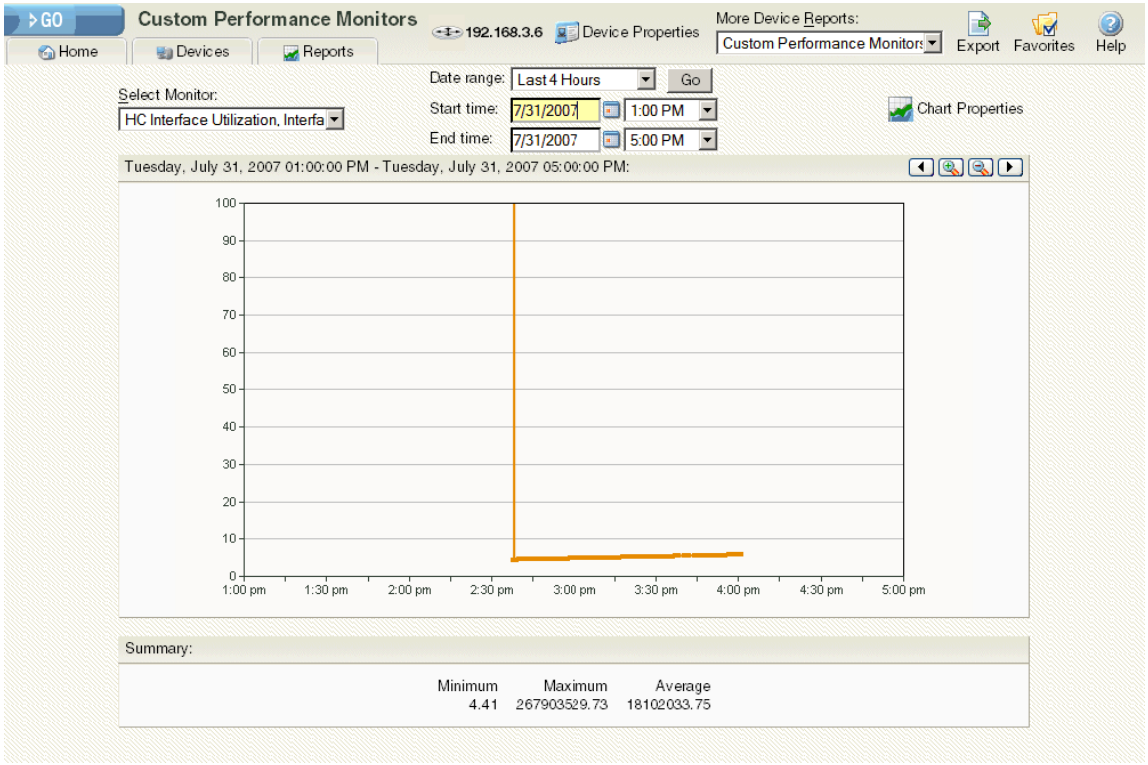
4 Click **OK**. The Device Properties dialog closes.

Step 4: View the data

1 From the main menu, select **Reports > Device**. The Device Reports category page opens.

2 Select **Custom Performance Monitors**. The Custom Performance Monitors report page opens.

3 If you have more than one custom performance monitor, verify that the correct monitor is listed under **Select Monitor**.



4 The graph now shows the total utilization of the interface as a percentage of the interface's total capacity. The numbers graphed are derived from the high capacity counters.

Tip: Since the graph shows a percentage, you may prefer to modify **Chart Properties** so that the graph uses a **Fixed Scale** of 0 to 100.

Script Example

```
// Monitoring High Capacity Counters with WhatsUp Gold v11 Code Example
var ifHighSpeed = Context.GetReferenceVariable("nIfHighSpeed");
var ifHCInOctets = Context.GetReferenceVariable("nIfHCInOctets");
var ifHCOutOctets = Context.GetReferenceVariable("nIfHCOutOctets");

if (ifHCInOctets == null ||  ifHCOutOctets == null || ifHighSpeed == null)
{
    // polling of reference variables failed.
    Context.SetResult(1, "Failed to poll this device.");
}
else
{
    // total bandwidth:
    var nTotalOctets = parseInt( ifHCInOctets) + parseInt(ifHCOutOctets);
    Context.LogMessage("Current polled value: " + nTotalOctets);

    // Get the current date. It will be used as a reference date for the SNMP polls.
    var oDate = new Date();
    var nPollDate = parseInt(oDate.getTime()); // get the date in millisec in an integer.

    // Retrieve the octets value and date of the last poll saved in a context variable:
    var nInOutOctetsMonitorPreviousPolledValue = parseInt(Context.GetProperty("nInOutOctetsMonitorPreviousPolledValue"));
    var nInOutOctetsMonitorPreviousPollDate =  parseInt(Context.GetProperty("nInOutOctetsMonitorPreviousPollDate"));
```

```
    // Save curent values for next time:
    Context.PutProperty("nInOutOctetsMonitorPreviousPolledValue", nTotalOctets)
    Context.PutProperty("nInOutOctetsMonitorPreviousPollDate", nPollDate);

    if (isNaN(nInOutOctetsMonitorPreviousPolledValue) || isNaN(nInOutOctetsMonitorPreviousPollDate))
    {
        // the context variable has never been set, this is the first time we are polling.
        Context.LogMessage("This monitor requires two polls.");
        Context.SetResult(0, "success");
    }
    else
    {
        // compute the bandwidth that was used between this poll and the previous poll
        var nIntervalSec =  (nPollDate - nInOutOctetsMonitorPreviousPollDate)/1000; // time since last poll in seconds

        var nCurrentBps = (nTotalOctets - nInOutOctetsMonitorPreviousPolledValue) * 8 / nIntervalSec;
        Context.LogMessage( "previous value = " + nInOutOctetsMonitorPreviousPolledValue);
        Context.LogMessage("difference: " + (nTotalOctets - nInOutOctetsMonitorPreviousPolledValue) + " bytes");
        Context.LogMessage("Interface Speed: " + ifHighSpeed * 1000000+ "bps");
        Context.LogMessage("time elapsed since last poll: " + nIntervalSec  + "s");
        Context.LogMessage("Current Bandwidth utilization: "+ nCurrentBps + "bps");

        // output the percent utilization:
        Context.SetValue(100 * nCurrentBps/(ifHighSpeed * 1000000));
    }
}
```