This document describes the new methods on how to publish CI QoS and Alarm messages which have been added to the SDK. This document assumes the reader has experience in writing Perl code and knowledge of the Nimsoft SDK.

# Configuration Item

The definition of Configuration Items (CI) must follow the new object model.

* QoS & Alarm data are associated with a Computer System.
* Unique identifier for Computer System (CS)
* Unique identifier for each Configuration Item (CI)
* For each CI you can monitor one or more aspects:
* Read speed, write speed for disks (CI Metrics)
* Every metric monitored has a one-to-one relationship to a CI
* Every CI has a one-to-one relationship to a CS
* All is stored in a new central repository (database tables, see below for more details)
* Create the CI in Perl
* This makes the association to the CS and the Type, this association has to be created first of all.

# CI Type & Metric Type Definition

## CI Type Definition

The available ci types are stored in the table CM\_CONFIGURATION\_ITEM\_DEFINITION.

|  |  |  |
| --- | --- | --- |
| ci\_type | ci\_parent | ci\_description |
| 1 | NULL | System |
| 1.1 | 1 | System.Disk |
| 1.1.1 | 1.1 | System.Disk.Filesystem |
| 1.1.2 | 1.1 | System.Disk.Partition |
| 1.10 | 1 | System.File |
| 1.10.1 | 1.10 | System.File.XML |

The typical base categories for ci types are:

* System (1)
* Network (2)
* Application (3)
* Database (4)
* Private (9)

Private is reserved for customer own CI type and Metric type definitions.

## Metric Type Definition

The available metric type definitions are stored in the table CM\_CONFIGURATION\_ITEM\_METRIC\_DEFINITION table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| met\_type | met\_description | unit\_type | ci\_type | multi\_select\_separator |
| 1.10:18 | File Size In Bytes | B | 1.10 | NULL |
| 1.10:19 | File Size In KB | KB | 1.10 | NULL |
| 1.10:20 | File Size In MB | MB | 1.10 | NULL |
| 1.10:21 | File Size In GB | GB | 1.10 | NULL |
| 1.10:22 | File Size In TB | TB | 1.10 | NULL |

The SDK expects for some methods the metric type (met\_type).

When creating CI’s the CI name is the name of the item being monitored, intended to distinguish the CI’s (e.g. files, disks, network interface, … ). It is this name which is being used to display in USM.

# CI Handle

## CI Type Definition

In order to send Alarms and QoS messages you must create a CI Handle. For this the following new methods have been added, both methods return a handle. If the record doesn’t exist it will be created by the method.

### ciOpenLocalDevice(ci\_type, ci\_name)

First parameter is the CI type, which is issued when you register a new CI type in the central repository.

Second parameter is the ci\_name which is a descriptive name of the CI and should be used together with the type parameter be unique for the CI's associated with a device.

E.g. $pCI = ciOpenLocalDevice("1.10","genesys\_filesize.txt");

### ciOpenRemoteDevice(ci\_type, ci\_name, target\_host)

First parameter is the CI type, which is issued when you register a new CI type in the central repository.

Second parameter is the ci\_name which is a descriptive name of the CI and should be used together with the type parameter be unique for the CI's associated with a device.

Third parameter is the target host; creating a handle with this method can result in the creation of a device (CM\_DEVICE) and possibly a new Computer System.

E.g. $pCI = ciOpenRemoteDevice("2.2.1","Echo","www.nimsoft.com");

## QoS & Alarms

## QoS messages

## In order to be able send QoS a remote or local handle has to be opened, as described in the section CI Handle. Next a QoS handle has to be created as one would normally do. Then a binding between the device handle and the QoS handle has to be made. Once this binding is made, you can send QoS messages the way you would normally do.

To bind the device handle with the QoS handle the following new method is used.

ciBindQoS(ci\_handle,qos\_handle,metric\_type)

Example:

$pCI = ciOpenLocalDevice("1.10","genesys\_filesize.txt");

$qos = nimQoSCreate("QOS\_FILESIZE",$source,$interval,-1);

ciBindQoS($pCI,$qos,"1.10.18");

nimQoSSendValue($qos,"<File Name>",42);

## Alarm messages

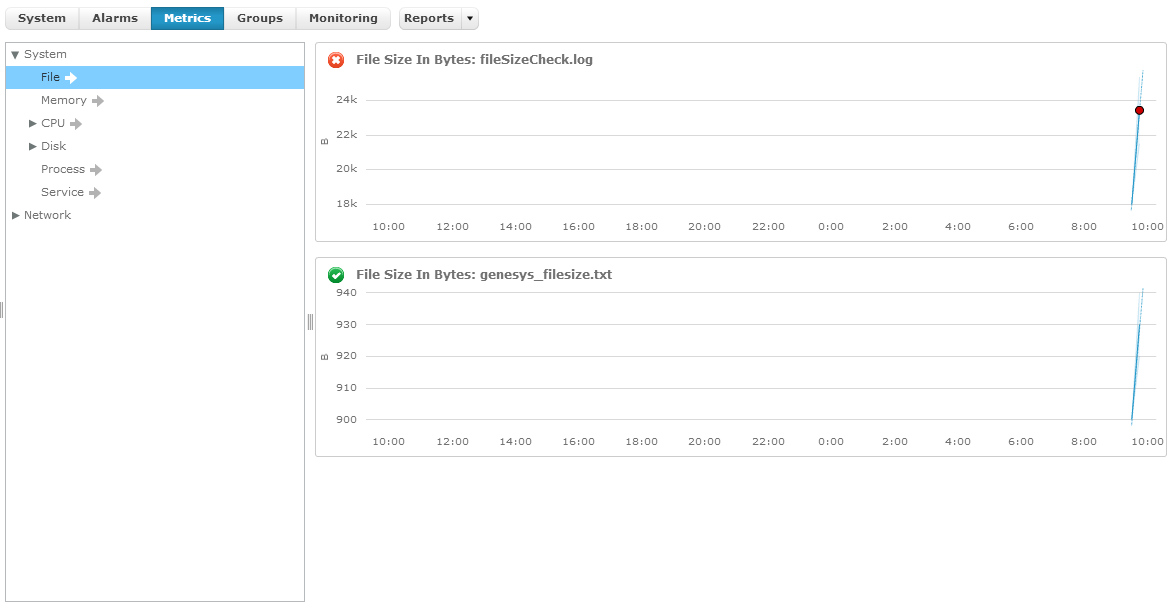
In order to be able send QoS a remote or local handle has to be opened, as described in the **section CI Handle**. Then you can send an alarm the way you normally would do but using the ciAlarm function, where the second parameter is the metric type for the alarm.

Example:

($rc,$alarmId) = ciAlarm($pCI,"1.10.18",5,"File $file size $filesize has exceeded threshold of $filethreshold",$token,$pds-data(),"1.10","$prgname $file threshold\_exceed","$source");

The usage of parameter $token is still unknown, use the code from the FileSizeCheck probe. For more details on the other parameters see perldoc.

## USM Screenshot



FileSizeCheck Probe

The source is hard coded in this probe, change accordingly. This probe has only been configured to run on windows operation systems.

!perl

#use lib "c:\\program files (x86)\\nimsoft\\perllib\\";

use Nimbus::API;

use NimBUS::CFG;

use Nimbus::Session;

use Nimbus::PDS;

my $prgname = "fileSizeCheck";

my $qosDefinition = 0;

my $filesize = 0;

my $source = "192.168.110.161";

my $next\_run = time();

my $alarmId = 0;

my $rc = 0;

my $config = "";

my $loglevel = 2;

my $logfile = "$prgname.log";

my $filename = "$prgname.txt";

my $interval = 60;

my $filethreshold = 2000;

###########################################################

# Command-set callback function(s), with parameter transfer

#

sub get\_size {

my ($hMsg,$arg1,$arg2,$arg3) = @\_;

my $reply = pdsCreate();

nimLog(2, "[get\_size] INFO: Sending request filesize $filesize");

pdsPut\_INT($reply,"filesize",$filesize);

nimSendReply($hMsg,0,$reply);

pdsDelete($reply);

}

###########################################################

# DoWork - function called by dispatcher on timeout

#

sub doWork {

my $now = time();

return if ($now < $next\_run);

$next\_run = $now + $interval;

my @myFiles = ( $filename, "fileSizeCheck.log");

foreach (@myFiles) {

monitor\_file($\_);

}

nimLog(0, "[doWork] INFO: Writing to file: $filename");

open (MYFILE, '>>genesys\_filesize.txt');

print MYFILE "A test for genesys\n";

close (MYFILE);

}

sub monitor\_file {

my $file = shift;

my $token = "cp#$prgname"; # usage of token is still unknown

my $pds = new Nimbus::PDS();

$pds->string("geheim","ach wie gut dass niemand weiss ...");

my $pCI = ciOpenLocalDevice ("1.10",$file);

nimLog(0, "[doWork] INFO: $source: $file: pCI: $pCI");

# My code

unless (-e $file) {

$filesize = -1;

($rc,$alarmId) = ciAlarm($pCI,"1.10:18",5,"File $file does not exist",$token, $pds->data(),"1.10","$prgname $file file\_exist","$source");

nimLog(2, "[doWork] ERROR: File $file does not exist ($alarmId)");

} else {

($rc,$alarmId) = ciAlarm($pCI,"1.10:18",0,"File $file does exist",$token, $pds->data(),"1.10","$prgname $file file\_exist","$source");

$filesize = -s $file;

if ($filesize >= $filethreshold) {

nimLog(0, "[doWork] INFO: Problem with file size: $file");

($rc,$alarmId) = ciAlarm($pCI,"1.10:18",5,"File $file size $filesize has exceeded threshold of $filethreshold",$token, $pds->data(),"1.10","$prgname $file threshold\_exceed","$source");

nimLog(2, "[doWork] ERROR: File $file size $filesize has exceeded threshold of $filethreshold ($alarmId)");

if ($file eq "genesys\_filesize.txt" ) {

nimLog(0, "[doWork] INFO: Resetting file: $file");

open (MYFILE, '>genesys\_filesize.txt');

print MYFILE "A test for genesys\n";

close (MYFILE);

}

} else {

($rc,$alarmId) = ciAlarm($pCI,"1.10:18",0,"File $file size $filesize is within threshold of $filethreshold",$token, $pds->data(),"1.10","$prgname $file threshold\_exceed","$source");

nimLog(2, "[doWork] INFO: File $file size $filesize is within threshold of $filethreshold ($alarmId)");

}

}

if (my $qos = nimQoSCreate("QOS\_FILESIZE",$source,$interval,-1)) {

ciBindQoS($pCI,$qos,"1.10:18");

if ($filesize < 0) {

nimQoSSendNull ($qos,$file);

} else {

nimQoSSendValueStdev($qos,$file,$filesize,0);

}

nimLog(0,"[doWork] INFO: Publish $file, $filesize");

ciUnBindQoS($pCI);

nimQoSFree($qos);

}

ciClose($pCI);

}

#######################################################################

# Service functions

#

sub restart {

}

sub timeout {

doWork();

}

###########################################################

# Signal handler - Ctrl-Break

#

sub ctrlc {

nimLog(0,"Got a control-C so am restarting");

exit;

}

###########################################################

# MAIN ENTRY

#

$SIG{INT} = \&ctrlc;

$config = Nimbus::CFG->new("$prgname.cfg");

$loglevel = $config->{setup}->{loglevel}|| 2;

$logfile = $config->{setup}->{logfile} || "$prgname.log";

$filename = $config->{setup}->{filename} || "$prgname.txt";

$interval = $config->{setup}->{interval} || 60;

$filethreshold = $config->{setup}->{file\_threshold} || 2000;

nimLogSet($logfile,$prgname,$loglevel,0);

nimLog(0,"----------------- Starting (pid: $$) ------------------");

nimLog(2, "[main] INFO: Config file: $prgname.cfg");

nimLog(2, "[main] INFO: log level: $loglevel");

nimLog(2, "[main] INFO: filename: $filename");

nimLog(2, "[main] INFO: interval: $interval");

nimLog(2, "[main] INFO: threshold: $filethreshold");

nimLog(2, "[main] INFO: Defining QoS definition");

# Send the QoS Definition

nimQoSSendDefinition ("QOS\_FILESIZE", # QOS Name

"QOS\_FILE", # QOS Group

"File size", # QOS Description

"Bytes","B"); # QOS Unit and Abbreviation

$sess = Nimbus::Session->new("$prgname");

$sess->setInfo($version,"Nimsoft Software AS");

if ($sess->server (NIMPORT\_ANY,\&timeout,\&restart)==0) {

$sess->addCallback ("get\_size");

}else {

nimLog(0,"unable to create server session");

exit(1);

}

nimLog(0,"Going to dispatch the probe");

$sess->dispatch();

exit;