Custom Probes and USM

From QA Wiki

This document describes how to ensure that your custom probe metrics and alarms appear under the correct device in USM.

Contents

- 1 A brief introduction to the NIS2 data model
  - 1.1 Configuration Items and Metrics
  - 1.2 How it all comes together in USM
- 2 SDK support for NIS2
  - 2.1 General process
  - 2.2 C SDK code
  - 2.3 Perl SDK code
  - 2.4 Java SDK code
  - 2.5 e2e_appmon
  - 2.6 Lua

A brief introduction to the NIS2 data model

The NIS2 (also sometimes called TNT2) data model was created with several goals in mind:

- Create a central database containing descriptions of every metric that the probes collect.
- Create a central inventory of devices being monitored.
- Create a central inventory of components of the devices being monitored (disks, network interfaces, etc.)
- Allow localization of metrics (descriptions and units).

The NIS2 data model consists of three basic types:

1. A device is an IP addressable system. A device entry is implicitly created when a configuration item is created.
2. A configuration item (CI) is a component of an IP addressable system. (A disk, for example.)
3. A metric is data that the probe collects that is associated with a CI. (For example, free space on a disk.)

Configuration Items and Metrics

Every alarm and QoS metric should be associated with a NIS2 CI and metric type ID so that USM can show it in the correct device view.

1. A CI (Configuration Item) represents the component being monitored. The component is intended to be a physical element (like a disk). Each CI must be associated with an IP-addressable system - this is one of the limitations of the NIS2 data model. The probe must set the following three values when a CI is created with the Nimsoft SDK:
    1. A CI type identifier that specifies the type of component. This is a dotted-decimal representation (for example: 1.1) with an English text representation stored in the database (1.1 = "System"."Disk"). The dotted decimal representation is localized. The localized text is contained in properties files that are shipped with wasp, and can also be updated separately using the wasp_language_pack package. The CI type IDs are defined in the CM_CONFIGURATION_ITEM_DEFINITION table.
    2. A CI name that represents the name of the component being monitored (for example, the name of a disk or ethernet interface: "eth0").
    3. A IP address to associate with the CI. This may be the IP address of the local system being monitored, or the IP address of a remote device.
2. A metric type ID represents the kind of measurement being collected. This ID is a single integer but is typically referenced using its fully-qualified path including the CI type, where it is the number after the colon (for example, the full path of 1.1:12 contains the CI type ID of 1.1 plus the metric type ID of 12, where the combination defines a unique kind of metric that can be collected: "System"."Disk":"Disk Free"). The metric type ID includes the metric unit, for example: 1.1:12 = "System"."Disk"."Disk Free" reported in GB and 1.1:13 = "System"."Disk"."Disk Free" reported in MB. The metric type IDs are defined in the CM_CONFIGURATION_ITEM_METRIC_DEFINITION table and units are defined in the CM_METRIC_UNIT table.

Creating CIs that represent the components being monitored is the substantial change required to use the NIS2 model. There are many CIs defined by Nimsoft in the CM_CONFIGURATION_ITEM_DEFINITION table of the NIS2 DB (the Nimsoft database typically named NimsoftSLM). Custom probe writers should use one of the provided CI definitions if one is applicable. If no existing CI definition can be used, a CI starting with 9 ("Private") may be used without contacting Nimsoft, or a range within the "Enterprise" or "Vendor" address spaces can be allocated through Nimsoft support or engineering. Allocating a range within the "Enterprise" or "Vendor" address space is recommended to avoid collisions in the "Private" address space (this is applicable in the case where two NMS environments need to be brought together - for example, a company merger where both companies are using the NMS product). Any additions to the CM_CONFIGURATION_ITEM_DEFINITION table (other than within the "Private" space) that are not coordinated through Nimsoft support or engineering may be overwritten during a product upgrade.

How it all comes together in USM

The SDK creates files in the niscache directory (under the Nimsoft installation directory) that represent devices, configuration items, and metrics as a side effect of calling the CI functions (ciOpenRemoteDevice) and using the ciAlarm and/or ciBindQoS functions. The discovery server periodically looks for these files and creates database entries based on the file content from each robot. QoS messages and alarms contain (in the message header) two new properties: dev_id (a link to the device) and met_id (a link to the metric). USM associates the alarms and metrics with the appropriate device based on these properties.

SDK support for NIS2

Probe code must be modified to use new API calls that create the NIS2 components and link the alarms and QoS messages that probes produce with those components.

General process

Probes need to follow the general process outlined below in order to submit QoS messages and/or alarms that will be properly linked to USM:

1. Create a CI (Configuration Item) that represents the component being monitored.
2. Link the CI and metric type ID to a QOS metric and/or alarm that is being delivered.
3. Post the QoS metric and/or alarm.
4. Unlink the CI and metric to free internal SDK resources.

C SDK code

```
CIHANDLE *h;

/* Create the CI and the device for a remote host being monitored */
h = ciOpenRemoteDevice(<CI type>, <Host name or IP address>, <CI name>)

/* or, create the CI and the device for the local system being monitored */
/* h = ciOpenLocalDevice(<CI type>,<CI name>) */

/* Use the new ciAlarm function to send an alarm associated with the CI and device created above
ciAlarm(h,<CI metric type>,…)

/* Associate the given QOSHANDLE (pointer to a QoS that has been created [code not shown]) with
ciBindQoS(h,<CI metric type>,<QOSHANDLE>, …)

/* Free the CI */
ciClose(h)
```

Perl SDK code

```
#!perl/bin/perl
use lib "perllib/";
use Nimbus::API;

$ENV{'NIM_ROOT'} = '/opt/nimsoft';

nimInit(0);

#
# NIS2 - Local device example
#
# This example shows how to create a CI representing
# a directory on the local system, and a metric representing
# the available space on that directory. A device that represents
# the local computer system where this probe is running is
# implicitly created by the ciOpenLocalDevice function.
#

# The name of the directory this probe is monitoring
my $dirName = "/var";

# The CI type is the dotted decimal type path (see the
# CM_CONFIGURATION_ITEM_DEFINITION table in the DB for a list of defined types).
my $dirCiType = "1.11"; # System.Directory
```

```
# The metric type is the kind of metric to collect for the configuration item
# (see CM_CONFIGURATION_ITEM_METRIC_DEFINITION for defined metric types).
my $dirMetric = "3"; # Directory Space in KB

# The (simulated) measurement value
my $dirSpace = 100; # Measurement - 100 KB remaining

# This creates a CI representing the "/var" directory on the local system.
my $hCI = ciOpenLocalDevice($dirCiType, $dirName);

# Send an alarm for the CI and metric ID
my ($rc,$szId) = ciAlarm($hCI, $dirMetric, 3, "$dirName space is low");

# Define a QoS metric for directory space
my $qosName = "QOS_DIRECTORY_SPACE";
my $qosGroup = "QOS_MACHINE";
my $qosDescr = "Directory Space";
my $qosUnit = "Kilobytes";
my $qosUnitAbbr = "KB";
my $qosInterval = 300;
my $qosSource = ""; # Nimsoft SDK will use the local host address

my $rc = nimQoSSendDefinition($qosName, $qosGroup, $qosDescr, $qosUnit, $qosUnitAbbr);
my $hQoS = nimQoSCreate($qosName, $qosSource, $qosInterval);

# Bind the CI to QoS (establish their relationship)
my $rc = ciBindQoS($hCI, $hQoS, $dirMetric);

# Send the QoS data point
my $rc = nimQoSSendValue($hQoS, $dirName, $dirSpace);

# Clean up
my $rc = ciUnBindQoS($hQoS);
my $rc = nimQoSFree($hQoS);
my $rc = ciClose($hCI);

nimEnd(0);
```

Java SDK code

Imports:

```
import com.nimsoft.nimbus.NimAlarm;
import com.nimsoft.nimbus.NimQoS;
import com.nimsoft.nimbus.ci.ConfigurationItem;
import com.nimsoft.nimbus.PDS;
```

Code:

```
// The ciType is the dotted decimal type path (see the
// CM_CONFIGURATION_ITEM_DEFINITION table in the DB for a list of defined types).
String ciType = "1.1"; // System.Disk

// The ciName is the instance name of a particular configuration item
// (for example, the name of a disk ("C:") or an ethernet interface ("eth0").
String ciName = "C:";

// The metricType is the kind of metric to collect for the configuration item
// (see CM_CONFIGURATION_ITEM_METRIC_DEFINITION for defined metric types).
String metricType = "12"; // Disk Free in GB

// ConfigurationItem holds most of the key information for the NIS2
// data model, except for the metric type.
// Create a ConfigurationItem for the local device.
ConfigurationItem ci = new ConfigurationItem(ciType, ciName);

// Create a ConfigurationItem for a remote device.
// ConfigurationItem ci = new ConfigurationItme(ciType, ciName, remoteHostName, remoteIpAddr);
// Create an alarm for the configuration item and metric type.
int severity = NimAlarm.NIML_CRITICAL;
int measuredValue = 5;
String message = "Not much free disk space (" + measuredValue + " < 10%)";
String subsystem = "1.1.1.1"; // alarm subsystems are defined in nas
String suppressionId = null;
String source = null; // null: use local host as the alarm source
NimAlarm alarm = new NimAlarm(severity, message, subsystem, suppressionId,
                              source, ci, metricType);
alarm.send();
alarm.close();

// Create a NimQoS object linked to the same ConfigurationItem and metric.
String qosName = "QOS_DISK_FREE";
String group = "QOS_MACHINE";
String description = "Disk Free Space in gigabytes";
String unit = "Gigabytes";
String shortUnit = "GB";
String target = ciName;
NimQoS metric = new NimQoS(ci, metricType, qosName);
metric.setDefinition(group, description, unit, shortUnit);
metric.setTarget(target);
metric.setValue(measuredValue);
metric.send();
metric.close();
```

e2e_appmon

```
include "NimBUS-functions.src"
' Create a Configuration Item (CI) to represent the overall page.
' First parameter ("3.21") is the CI type ("Application"."E2E").
' Second parameter ("Nimsoft Site") is the name of the CI.
' Third parameter ("") is the remote device IP (empty means that the metric will be
'                          attached to the local robot device).
' Fourth parameter ("7") is the metric type ID number.
'
' USM contains special logic for "3.21" ("Application"."E2E") so that
' the name of each CI defines a child element in the tree. The CI name
' should therefore be set to the script name, and each script will then
' appear as a separate tree node.
nimSetCi("3.21", "Nimsoft Site", "", "7")

' nimInit must be called after nimSetCi
nimInit()

StartBrowser("IE", "www.nimsoft.com", 3)

' USM will sort the steps of a transaction in alphabetical order.
' Starting the target with the step number ensures that metrics
' will appear in the right order.
target$= "Step 1: Support"
nimQoSStart()


UsePage("ITSM + Service Desk + IT Monitoring: Better Together - CA Nimsoft")
        ClickHTMLElement("A[INNERTEXT= 'Support',INDEX='3']")

nimQoSStop()
nimQoSSendTimer(target$)

' This call is the same as above except that the metric is incremented
' to provide a unique identifier in the DB.
nimSetCi("3.21", "Nimsoft Site", "", "8")
nimInit()
target$= "Step 2: Downloads"
nimQoSStart()

UsePage("ITSM and ITIL-based Support and Service That You Can Count On - Nimsoft")
        ClickHTMLElement("A[INNERTEXT= 'Downloads',INDEX='2']")
```

```
nimQoSStop()
nimQoSSendTimer(target$)

CloseWindow("IEXPLORE.EXE|IEFrame|Nimsoft - Windows Internet Explorer",1)

CloseWindow("IEXPLORE.EXE|IEFrame|ITSM and ITIL-based Support and Service That You Can Count On

nimEnd()
```

Lua

The Lua SDK does not currently support the NIS2 functions.

Retrieved from "http://wiki.dev.fco/index.php/Custom_Probes_and_USM"

- This page was last modified on 6 February 2013, at 00:08.