CA Unified Infrastructure Management

Probe Software Developer Kit Guide

v2.1

ca technologies

Legal Notices

This online help system (the "System") is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This System may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This System is confidential and proprietary information of CA and protected by the copyright laws of the United States and international treaties. This System may not be disclosed by you or used for any purpose other than as may be permitted in a separate agreement between you and CA governing your use of the CA software to which the System relates (the “CA Software”). Such agreement is not modified in any way by the terms of this notice.

Notwithstanding the foregoing, if you are a licensed user of the CA Software you may make one copy of the System for internal use by you and your employees, provided that all CA copyright notices and legends are affixed to the reproduced copy.

The right to make a copy of the System is limited to the period during which the license for the CA Software remains in full force and effect. Should the license terminate for any reason, it shall be your responsibility to certify in writing to CA that all copies and partial copies of the System have been destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS SYSTEM “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS SYSTEM, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The manufacturer of this System is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Legal information on third-party and public domain software used in the Nimsoft Monitor solution is documented in Nimsoft Monitor Third-Party Licenses and Terms of Use *(*http://docs.nimsoft.com/prodhelp/en_US/Library/Legal.html*).*

Contact CA

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

Send comments or questions about CA Technologies Nimsoft product documentation to nimsoft.techpubs@ca.com.

To provide feedback about general CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Revision History 7

Chapter 1: Overview of Monitoring Probes 9

Probe Purpose 9
Inventory Monitoring 10
Terminology 10

Chapter 2: Getting Started 15

Who Should Use the Probe-SDK 15
Probe-SDK Package 15
Javadoc and Help ..... **Error! Bookmark not defined.**
Programming Language 16
Development Environment 16
Probe Examples 16
Deployment Environment 18

Chapter 3: Design Considerations 19

Factors that Influence Probe Design 19
CI Modeling 20
- Management Types 21
- CI Model with No Organization 21
- CI Model with Device Components 22
- CI Model with Logical Containers 22

Display of Probe Inventory 23
- Local Probe Inventory 24
- Remote Probe Inventory 25

Probe Complexity 25
Inventory Modeling 26
QoS Metric Type Selection 27
- TNT2 Data Model 27
- CIs and Metrics 27
- Metric Type Creation 27
- Predefined Metric Types 28
- Metric Type Registration 28

Identifiers 28

Chapter 4: Verify Your Build Environment 29

Build a Probe 29
(Optional) Open the Project in an IDE 30
Deploy Probe 30
    Deploy in Admin Console 30
    Deploy in Infrastructure Manager 31

Chapter 5: Create a Probe 33

Files for Probe Development 33
    Probe_schema.cfg 33
    pom.xml 35
    <probe>.cfx 35
    <probe>.pkg 37
Add Libraries and Other Dependencies 40
(Optional) Update NAS Subsystem IDs 40
Debugging a Probe 41

Chapter 6: Example 43

Revision History

Date	Description	Version
June 2015	Simplified the probe development process and probe examples. Reduced the size of the Probe-SDK.	V2.1.0
March 2015	Removed dependency issues with the CA genzip plugin	v2.0.1
January 2015	This version contains updates for building probes on an 8.1 or newer CA UIM platform.	v2.0.0
October 2014	Corrected handling of package dependencies.	v1.1.4
October 2014	Added the capability to build probes with Maven off-line.	v1.1.3
October 2014	Corrected versioning of product dependency jar files.	v1.1.2
September 2014	Initial release	v1.1.0

Chapter 1: Overview of Monitoring Probes

This section provides an overview of CA Unified Infrastructure Management (CA UIM) probes, and describes concepts for developing a monitoring probe.

Note: This version supports probe builds for CA UIM version 8.1 or newer.

Probe Purpose

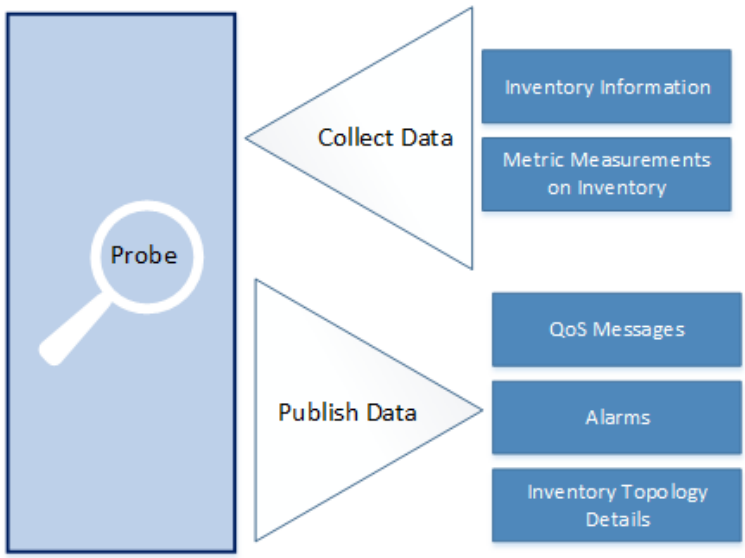
A monitoring probe is a specialized piece of software that collects and publishes monitoring data.

The primary function of a probe is to identify devices and components, and publish Quality of Service (QoS) metrics on the identified devices and components. You can configure the targeted devices, components, and the publishing of QoS metrics within a probe.

Probes can also be configured to emit alarms relative to various QoS thresholds. Some probes only publish the QoS metrics and rely on external services to configure and process thresholds. Other probes provide their own threshold configuration and processing to emit alarms.

Another function of a probe is to publish the relationships of devices and components within an environment. These relationships create a topology (or hierarchy) for the probe inventory. When the probe publishes its inventory, the inventory is added to the device and component topology which represents the infrastructure under management.

The following figure shows the data that a probe collects and the data a probe publishes.



Inventory Monitoring

Various probe inventories contribute to the broader discovery of infrastructure which provides the solution services you need. You can use CA UIM and the discovered inventories to configure probes in various ways to present and report the monitored information.

The probe inventory can contain many types of hardware or software components. Some examples are computer systems, networking devices, storage devices, hypervisors, virtual machines, CPUs, memory, processes, files, databases, or webservers. You can model almost anything as an inventory element with associated QoS metrics.

Terminology

The following terms are used throughout this document.

Element Type

The Probe-SDK facilitates the definition of various types of elements. These element types model various components and devices which creates the probe inventory. Element types are defined in the probe_schema.cfg file along with the QoS metric types. Java classes can be generated to represent the element type and provide programmatic convenience for working with element instances. The definition of new element types is facilitated by extending standardized base element types.

Element

An element is an instance of an Element Type. Element objects model various kinds of components or devices to create a probe inventory. For example, computer systems, virtual machines, hard disks, and processes are all modeled as a topology of elements.

Configuration Item (CI)

A CI represents an instance of a CI type with the associated metrics published by the probe.

CI Type ID (CI Type)

A CI type represents a group of related metric types. For example, the CI type for Disk might contain metrics for throughput, capacity, and free space. A numeric value represents a CI type. For example, the CI type for System.Disk is 1.1.

Note: CIs are also known as metric families where a group of metrics relates to multiple component types.

Example:
Throughput, Capacity, and Free Space are the available measurements for a disk. Each of these measurements would have the representative CI type of System.Disk.

Metric Item (MI)

A MI represents an instance of a metric type which is a QoS measurement for a specific CI.

Example:
The Throughput measurement for the System.Disk CI contains the MIs Read Throughput, Write Throughput, and Total Throughput.

Metric

Quality of Service (QoS) measurements are captured by using metrics. Metrics and alarms must be defined for the probe monitoring target.

Metric Type ID (Metric Type)

A metric type specifies a measurement type that is available for a CI type. A metric type is a numeric string that represents a CI type and the associated MI type.

The string 1.1:29 is an example of an association between a CI type and a metric. The number 1.1 before the colon is the CI type for System.Disk, and 29 is associated with the metric type associated with the CI. For example, Read Throughput.

Example:
The System.Disk CI type Throughput measurement contains the MIs Read Throughput, Write Throughput, and Total Throughput. The metric types are:

- 1.1:29 - System.Disk:Read Throughput
- 1.1:30 - System.Disk:Write Throughput
- 1.1:31 - System.Disk:Total Throughput

Note: Metric types are defined and managed in database tables with their associated CI type, description, and units. Element types are associated with metric types to model a component and provide the desired QoS data.

Component

Components are usually part of a larger device and do not have an IP address. The subcomponents of a device are identified relative to an IP device.

Note: A component is modeled by an element. The infrastructure under management provides services where the solution is composed of many components. A component is often thought to be some physical part of a larger device, but running software can also be considered a component. The word element is frequently substituted for component.

Device

A device is a high-level component identifiable through an IP address. Some examples of devices are computer systems such as servers and networking components.

Device ID

A device ID is an encoded identifier for a device that you use in post processing and database tables.

Note: A device is also associated with a human readable identifier. The Probe-SDK process automatically generates device IDs.

Origin

The origin is one of several properties set on a QoS message. The default origin is the hub, but you can change this property to any value through the robot.

Note: A QoS message has properties for the QoS measurement value and timestamp, origin, source, and target. A Device ID and a metric ID are also associated with a QoS message.

Metric ID

A metric ID is an encoded identifier for a metric measurement on an element or component instance. The Probe-SDK process automatically generates metric IDs.

Example:
If a probe is sending status QoS messages on a specific hard disk, the probe uses the same metric ID in the messages.

Source

The source is an IP addressable system or IP device.

The QoS message includes the source when a probe publishes a QoS message for a metric value on a component. If the source is an IP device, the source is the IP device or the parent IP device of the component.

Target

The target is the element or component where the probe collects the metric measurement.

Profile

You use profiles to configure what inventory a probe monitors and at what frequency. A scheduler runs profiles at defined intervals. The intervals result in the collection and publishing of QoS and inventory data.

Note: A profile is sometimes called a resource on remote probe configurations.

QoS Definition Message

A QoS definition message provides additional overview information for each type of QoS message. A probe publishes a QoS definition message on the bus before publishing its associated QoS Message type. The Probe-SDK encapsulates these details away from the developer.

QoS Message

A published QoS message has a measured value, at a particular point in time, for a device or component. All QoS messages include origin, source, target, metric ID, device ID, and any other additional properties.

For example, a CPU Usage measured value of 95 percent, at the time 10:30 am, core #2 of CPU #4 in a server.

QoS Name

The QoS metric name that follows the format of QOS_<APPLICATION/PROBE_NAME>_<UNIQUE_IDENTIFIER>. The entire QoS name is capitalized with no spaces. For example, QOS_CLOUD1_MONITORNAME.

Note: A QosName and MetricType are associated with an element type. This association provides sufficient definition to publish QoS message measurements for an element instance.

Monitor

Monitors publish QoS data, process thresholds, and publish alarms. Monitors are configured through the probe UI or centralized services.

The most basic monitor enables the publishing of QoS data and is called a QoS monitor, or a QoS only monitor. A QoS monitor type is defined primarily by a metric type, QoS name, and an identifying key (in the probe_schema.cfg), and is associated with an element type. QoS monitor instances are configured to apply to element instances. QoS monitor configurations are generally saved in the <probe_name>.cfg file.

Various threshold and alarm monitors can be applied to a QoS measurement. The probe can provide these monitors, but the trend is to configure monitors through centralized services.

Chapter 2: Getting Started

The Probe Software Developer Kit (Probe-SDK) provides examples of monitoring probes that you can study, or can copy and modify to create new probes. This section describes the content of the Probe-SDK and provides guidelines on how to use the Probe-SDK.

Who Should Use the Probe-SDK

The Probe-SDK is for developers who want to create a monitoring probe that is configured through Admin Console and any other CA UIM interfaces. The Admin Console application allows you to configure probe profiles, to observe the collected probe inventory, and configure related QoS monitors.

Note: The Admin Console is a thin web-based interface that can be used to manage and maintain your CA UIM hubs, robots, and probes.

The Probe-SDK also allows you to develop a probe that can leverage common CA UIM functionality. Some examples of this common functionality are dynamic alarm thresholds and static thresholds. The Probe-SDK also provides a consistent probe architecture that you can easily maintain over time.

Probe-SDK Package

Obtain a copy of the probe-sdk-<*version*>.zip package from the support Downloads page (http://support.nimsoft.com).

Important! The probe-sdk-<*version*>.zip package is not a standard probe package. Do not attempt to load this file through Admin Console or Infrastructure Manager.

The probe-sdk-<*version*>.zip package contains the following components:

- **docs directory** – This directory contains a readme file, javadocs, and a list of standard metrics (MetricTypesWithUnits.SpreadSheetOfMostCommonTypesForReuse.TNT2.xlsx and MetricTypesWithUnits.AllRegisteredTypes.TNT2.csv).
- **archetypes directory** – This directory contains various examples of probes for study and to use as starting templates.
- **libs directory** - This directory contains the probe-api jars and the required Maven plugins. This content is all that you need for development on your local system.
- **Install scripts** – Scripts to install the required libraries and archetypes into your Maven repository.

Note: Both .bat and .sh scripts are provided for your convenience. Run .bat scripts in a Windows environment and .sh scripts in a Linux environment.

Programming Language

For this guide, the programming language that is used is Java.

Development Environment

Windows or Linux development environments are supported. Verify that your development environment is set up before you begin any type of development.

To set up your development environment:

- Obtain and unzip the probe-sdk-<*version*>.zip package. This creates a probe-sdk directory.
- Install Maven 3.0 or later. Ensure that maven is in your path by running mvn –version.
- Install Java SE SDK 7.
- (Optional) Install an integrated development environment (IDE) that supports Java such as Eclipse or NetBeans.

Probe Examples

Examples of probes are provided with many of the design elements you must consider when creating a probe. Select the appropriate probe example to copy and use as a starting point for your probe development process.

The following probe examples are in the Probe-SDK examples directory.

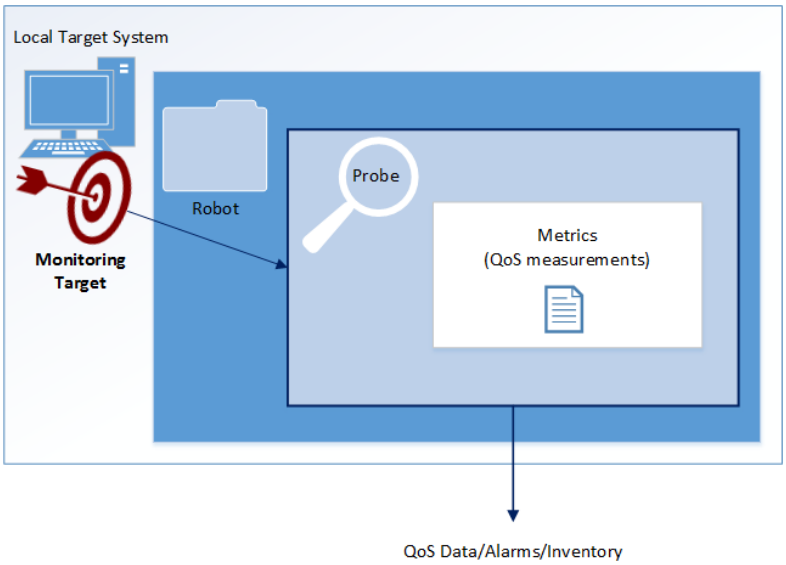
- probe_sdk_hello_world – An empty probe container that contains comments describing sections of code.

  You can deploy and run this probe, but you must add topology and metrics information to create a meaningful probe. Choose this probe if you need a template to build a new probe with no pre-existing topology or metrics.

  Note: It might be easier to start the development process by modifying a copy of an example with more content.

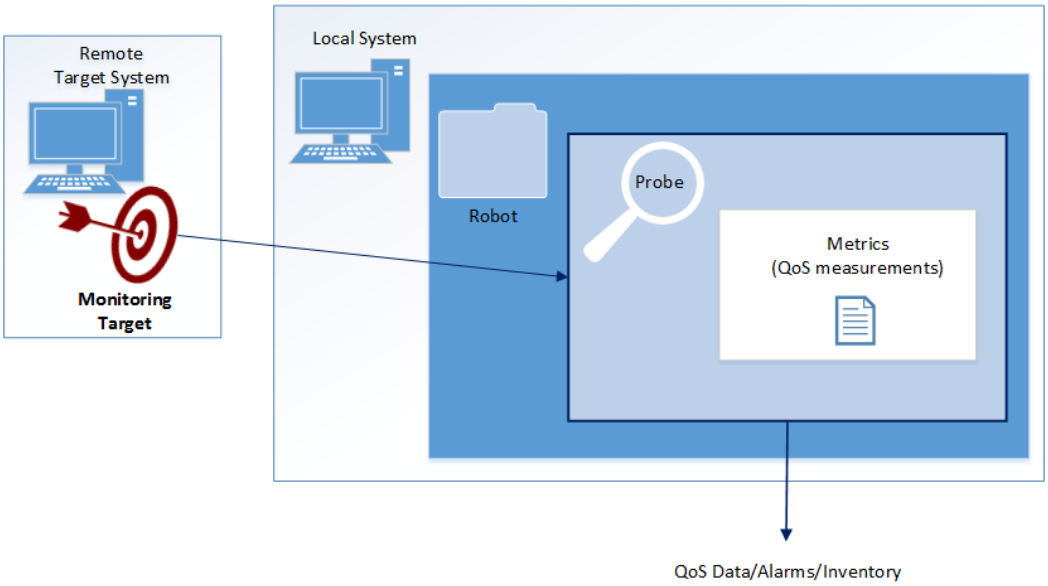
- probe_sdk_local_dirscan – An example of a local probe. This simple probe monitors a specific directory which you identify in the configuration of a profile.

  Local probes are installed on the device to be monitored. The following figure shows the minimal architecture components that a local probe requires.



- probe_sdk_mock_vm_host – An example of a remote probe with probe oriented bulk configuration. This probe contains a mock connection to a hypervisor. This probe monitors the infrastructure of hosts and virtual machines.

Remote probes are installed on a different device and perform monitoring across a network. The following figure shows the minimal architecture components that a remote probe requires.



Deployment Environment

In order to deploy and test a probe, you need:

- A full CA UIM installation to deploy the probe and collect probe data.
- Unified Management Portal (UMP) to verify that the probe works.
- A system that is installed with a robot.

  Note: The software can be co-located with CA UIM for initial testing. You must move the software components to a more distributed model as the test load increases.

Chapter 3: Design Considerations

This section describes how your decisions during development can affect the probe web user interface (UI) usability and behavior.

Factors that Influence Probe Design

Consider the following aspects of probe design:

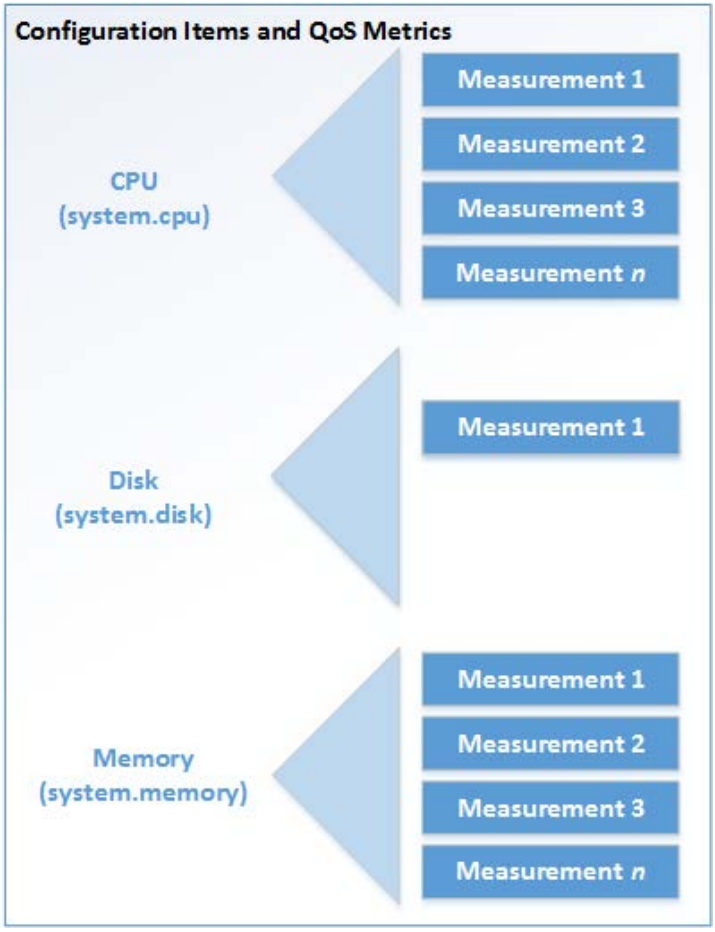
- Model the desired inventory devices and components as elements with the desired QoS metrics.
  - Decide on a consistent method to identify element instances.
  - Select which Java base elements to extend to represent a given device or component.
  - Select the desired QoS metric monitors to go with an element.
- Determine how the probe collects the inventory and metrics.
  - Identify what commands the probe must run for which components or metrics at what frequencies.
  - Determine what parsing must be done.
  - Identify what types of connectivity you need for your target devices.
- Design the profiles and their configuration.
  - Identify what profile configuration parameters you need.
  - Decide if your profile configuration parameters can be managed with a single profile type, or if you need multiple profile types.
  - Determine if your profiles are configured through probe configuration parameters, automatic discovery, or a combination of configuration parameters and automatic discovery.
- Design for performance, scalability, and robustness.
  - Identify any performance and scale issues that can exist with large inventories with lots of metrics.
  - Determine if concurrent processing is necessary to meet the desired collection rates. Each profile instance runs its own thread during collection, but a profile can require multiple threads to increase concurrency.
  - Verify that the inquiry load placed on the devices being monitoring remains within acceptable levels.
  - Verify that the overall probe footprint and CPU load remains at acceptable levels.

- Verify that the internet bandwidth consumed by monitoring inquiries and data transfers remains within an acceptable range.
- Verify that the size of the probe.cfg file remains at an acceptable size for large configurations.

CI Modeling

How you choose to model metrics in the probe_schema.cfg file impacts how the metrics appear in the probe UI. Model your metrics to create an efficient configuration interface.

The following figure shows an example of CIs that can contain one or many metrics.



Management Types

In general, the structure of the CIs you create must match the structure of the target you want to monitor. Consider one or more of the following concepts to organize your metrics.

IP Devices - Elements that exist within a network.

Hardware Components - Elements that exist within a device.

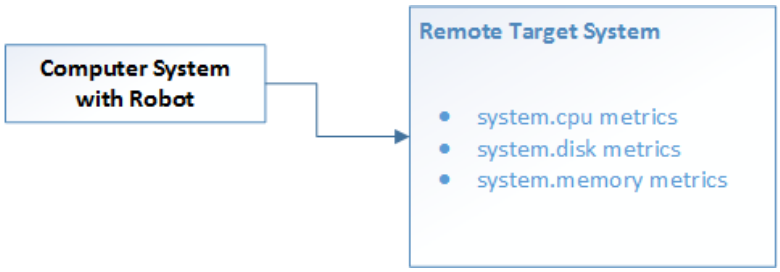
Software Components – Elements that exist within a software application.

Logical Containers – Represent groups of related elements.

Folders – Generic container for elements that do not fall within any of the other management types.

CI Model with No Organization

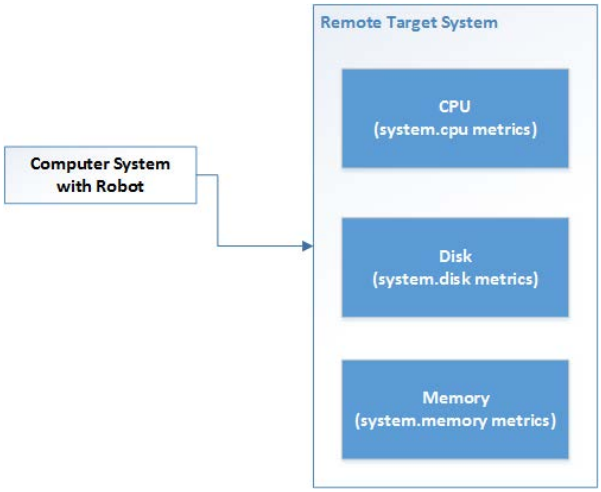
The following figure shows an example of a CI model with no components to organize the QoS measurements.



In this example, the probe configuration UI displays a device node and a list of all metrics. This model is efficient only if there are a few metrics collecting device data.

CI Model with Device Components

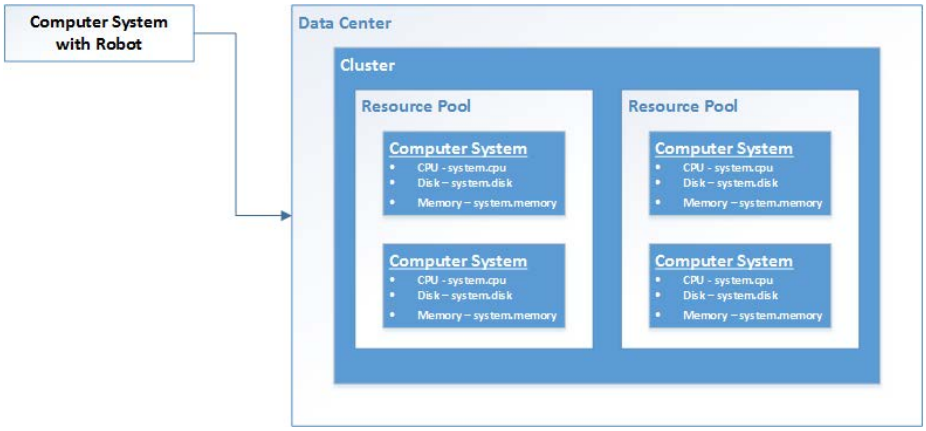
If you want to collect individual QoS metrics on multiple components within the target device, you must define multiple CI elements for the probe. The following figure shows an example of a CI model with organizational components for the QoS measurements.



In this example, the probe configuration UI displays a node for the device and child nodes for CPU, Disk, and Memory for the associated metrics. By adding this organization, the probe administrator can quickly locate the appropriate metrics by the CI type. This example shows the hierarchy of hardware components within a device. Alternately you can use this hierarchical organization to organize metrics for IP devices and software components.

CI Model with Logical Containers

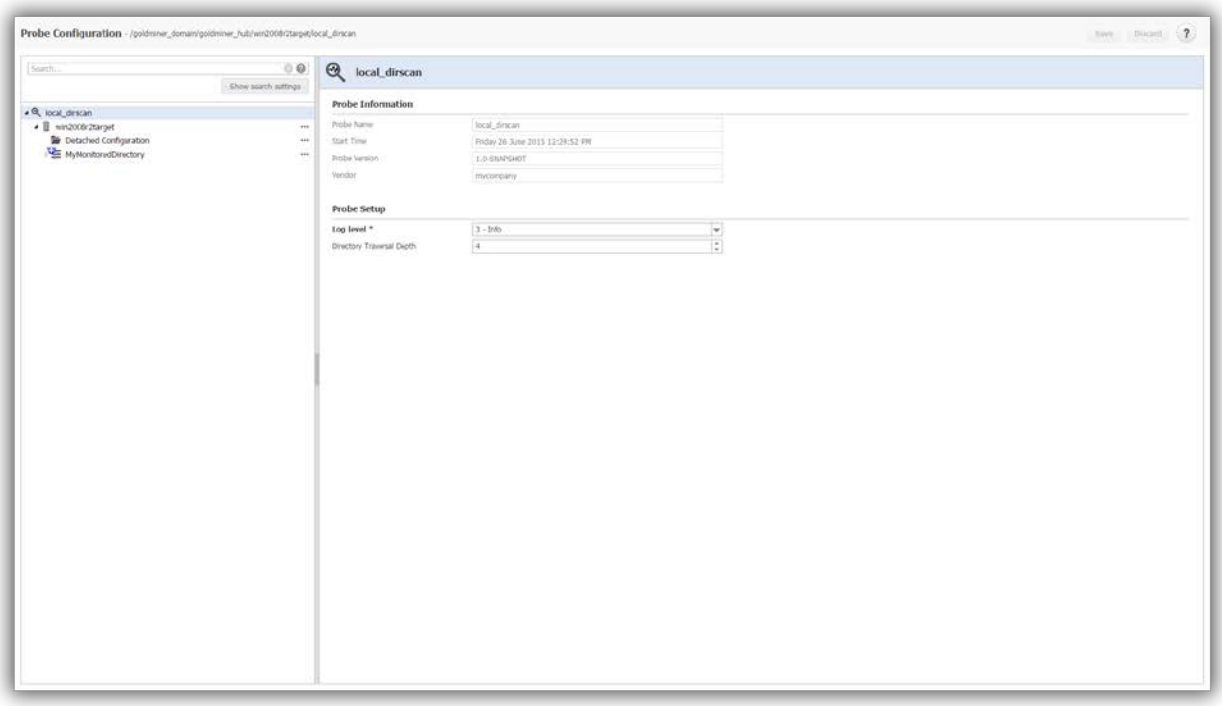
The following figure shows an example of a CI model with organizational components representing logical containers for the QoS measurements in addition to device components.



In this example, the probe configuration UI displays a node for a datacenter and a nested set of child nodes for the associated clusters, resource pools, and computer systems. Each of the computer system nodes contain the metrics for CPU, Disk, and Memory. This example only contains metrics for individual computer systems, but more metrics could be added to monitor elements throughout the hierarchy. For example, you could add a metric to monitor the load on a resource pool.

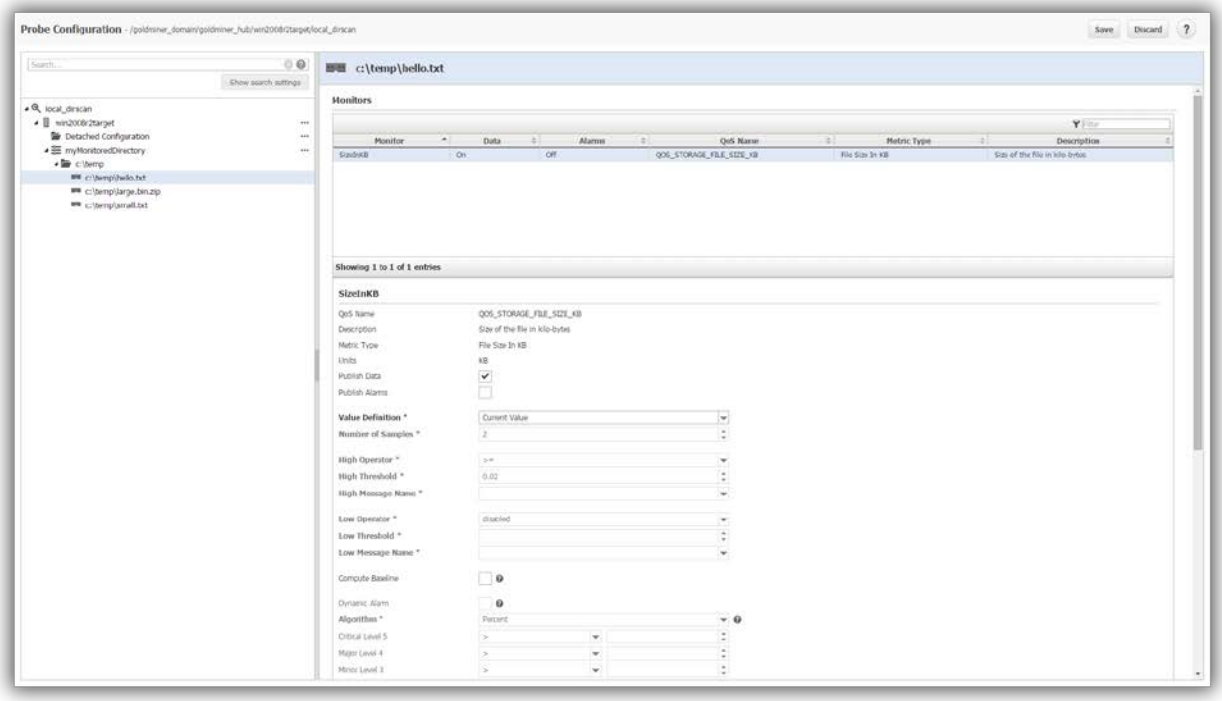
Display of Probe Inventory

This section provides some examples describing how the probe configuration UI displays the probe inventory. The following figure shows the probe configuration UI for the probe_sdk_local_dirscan probe. This probe is included in the Probe-SDK as an example.



The left pane shows an expanded tree view with the probe at the top of the hierarchy. The right pane shows the configuration properties that are associated with the selected tree element. In this example, the right pane shows the configuration parameters for a profile. A profile is run at a specified interval to collect inventory and metrics according to these configuration parameters.

The following figure shows the probe_sdk_local_dirscan probe UI with a selected inventory file element.

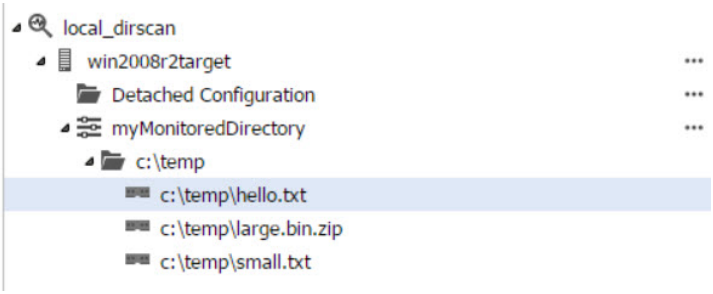


The right pane shows the QoS metric monitors that are associated with the inventory file element.

Local Probe Inventory

The probe_sdk_local_dirscan probe is a local probe. Local probes monitor components on the computer system where the probe runs with a controlling robot. The inventory for local probes shows the local computer system between the probe and its profiles. This structure indicates that the local computer system is the relevant IP device.

The following figure shows an example of the probe_sdk_local_dirscan probe inventory.



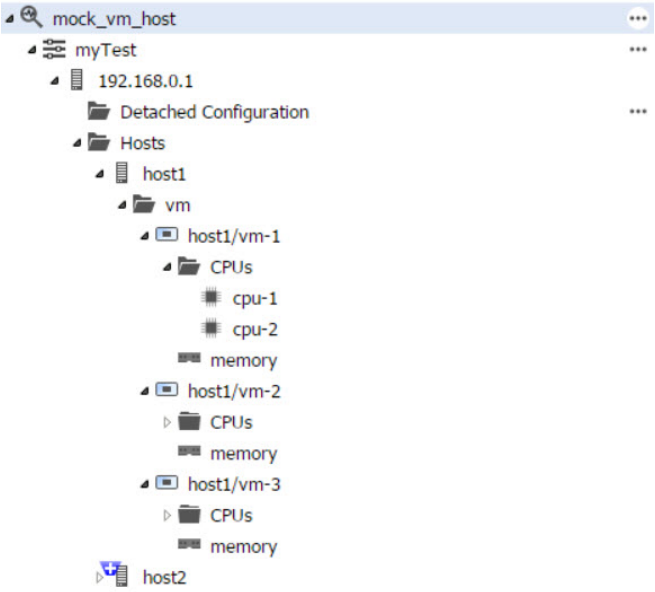
Inventory is organized relative to an IP device. An IP device is a device identifiable and accessible through the Internet. For a local probe, the local computer system is the target IP device from which the inventory is discovered. The inventory is shown in the tree view to highlight this architecture.

Note: You can use folders to organize groups of elements within the inventory tree, or show an alternate view. In the previous example, you could add a folder with the name Over 10 MB. This folder provides an alternate view of the inventory file elements for files larger than 10 MB.

Remote Probe Inventory

Remote probes typically use some form of internet communication to the external devices to collect the desired inventory and metrics. The profiles for remote probes specify a remote IP device target with credentials in their configuration parameters.

The following figure shows the probe_sdk_mock_vm_host probe inventory. This probe is included in the Probe-SDK as an example.



In the tree view, remote profiles are shown between the probe and their target IP device. The collected inventory for the target IP device is shown below the IP device node.

Probe Complexity

The probe design can be relatively simple or complex due to various design factors.

Simple Probes:

- Have simple inventory models with a few types of elements and a few associated metrics.
- Have a single profile type with the name RESOURCE, and a simple set of configuration parameters.
- Collect both inventory and metrics on the same interval for the profile instance.

Complex Probes:

- Have more complex inventory models with more element types and metrics.
- Have multiple profile types with advanced configuration parameters and actions.
- Often collect metrics at a higher frequency than inventory and therefore require different profile intervals and commands.
- Have performance and scale challenges for probes monitoring large inventories with lots of QoS metrics.

Inventory Modeling

You model inventory devices and components as Java elements with associated QoS metrics. You select elements from a range of Java base elements which have associated attributes. Attributes are fixed properties describing an element and metrics are changing properties made available as QoS monitors. You use relationships to relate element instances in a hierarchical manner, but cyclical graphs are possible.

When you model a probe, your first challenge is to select the base element which best represents the device or component type. You can find standardized base elements representing the most common devices and components in the java com.nimsoft.probe.framework.devkit.inventory package. Components use derivations from element, and devices use derivations from IPDeviceElement. These base elements are associated with standardized attributes.

Note: These standardized base element types with attributes are useful for the overall CA UIM topological modeling of the infrastructure. Related instances of base elements are used to model the topological graph for infrastructure. These standardized base types have standardized attributes. Metrics are not considered a part of the topology. This means base elements do not have predefined metrics.

A new element definition (ElementDef) is created for each new component or device type. Each Element instance has an ElementDef reference. This instance provides the flexibility of runtime definition and extension. The base element types encapsulate this instance for convenience, and custom component and device classes are often generated which extend the base elements.

QoS Metric Type Selection

You must select the appropriate QoS metric type to associate with the device or component type. All metric types are predefined and registered in a database using the TNT2 data model. This data model provides a centralized repository with localized descriptions of all metrics and units that you use when developing a probe with the Probe-SDK.

TNT2 Data Model

The TNT2 data model is a centralized repository that contains metric descriptions, an inventory of devices and device components, and localizable metric descriptions and units. The TNT2 Data Model is sometimes given the name NIS2.

The TNT2 data model contains three basic types:

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

CIs and Metrics

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

Metric Type Creation

Each metric type is associated with the element definition using a key name and a QoS name. The probe_sdk_local_dirscan and probe_sdk_mock_vm_host examples demonstrate this process.

For example, 1.1:10 is a metric type. The 1.1 indicates that this metric type is for the CI System.Disk and the 10 specifies the measurement for disk space used in MB. The CI type specifies a group of metrics. The 10 specifies the metric which has a localizable description and units. When associated with an element definition, this metric type might use UsageInMB as the key and QOS_STORAGE_DISK_USAGE_MB.

Note: The original concept for the CI type was that metric type grouping was based on component types. In current probe development, the CI is considered a group or family of metrics applicable to a type of component or a type of measurement.

Predefined Metric Types

A spreadsheet in the docs directory with the name `MetricTypesWithUnits.SpreadSheetOfMostCommonTypesForReuse.TNT2.xlsx` provides a quick reference to some of the metric types for general reuse.

Metric Type Registration

Metric type definitions are registered in the CM_CONFIGURATION_ITEM_METRIC_DEFINITION, CM_CONFIGURATION_ITEM_DEFINITION, and CM_METRIC_UNIT tables. These entries are required for proper consumption and correlation by CA UIM services for reporting and configuration.

All TNT2 entries that are intended for public use have gone through an approval process. Probes that are developed by CA or MarketPlace partners are both examples of public use. The TNT2 database has several thousand entries. Many entries are meant to be generic, and some probe entries are very specific. Use existing entries as much as possible.

Identifiers

Instances of elements must be uniquely and consistently identified for monitoring. Use GUIDS whenever they are available. However, GUIDs are often not available. You can use IP device elements because they are identifiable through their IP addressing information. Subcomponents of IP device elements which do not have independently unique identifiers are identified relative to their IP device elements.

IP device elements are used for correlation by the Discovery Server for the presentation of the infrastructure under management.

Elements often have attributes for serial numbers, MAC addresses, or other hardware identifiers. When available, set these attributes to help with overall correlation of QoS data. Attributes are especially useful when two probes are monitoring the same component, but might have different specifications for the primary identifier.

Chapter 4: Verify Your Build Environment

Before you attempt to modify the contents of a probe, verify that your environment is correctly set up. This section describes how to build and deploy a probe template.

The following figure shows an overview of build environment verification process.

How to Verify Your Build Environment

Developer

1. Build a Probe
2. (Optional) Open the Project in an IDE
3. Deploy Probe

Build a Probe

Install the required libs and project details to build a probe.

Follow these steps:

1. Open a command prompt.
2. Run: **install.bat –all**

   Important! You only need to run this command if this is the first time you are installing the probe-sdk-*<version>*.zip package.

   The command installs the libraries and archetypes into the Maven repository.
3. Create a directory in the location you want to use to create the probe.
   Example: mkdir c:\MyProbes
4. Go to the directory.
   Example: cd c:\MyProbes
5. Run: **mvn archetype:generate -DarchetypeCatalog=local**
   The software displays a series of prompts to install the probe.
6. Provide the following information:
   - Select an archtype. This is the example probe you want to use as a template.
   - Enter a groupID. For example, com.mycompany.

- Enter an artifactId. For example, MyDirscanProbe.
- Enter a probe version. Press return to accept the default value.
- Enter a package name. Press return to accept the default value.

7. Confirm your settings.
   The software creates all of the required files for the probe in the probe directory.
8. Go to the probe directory.
   Example: cd c:\MyProbes\MyDirscanProbe
9. Run: **mvn clean install**
   The command builds the final probe package. The probe package will be found in the probe directory.
   Example: c:\temp\MyDirscanProbe\target\MyDirscanProbe.zip

(Optional) Open the Project in an IDE

You can open a probe project in an IDE such as Eclipse with the project pom.xml file in the probe directory. You can also use the probe pom.xml file to build the probe jar file with Maven. The probe project is a modular project with a dependency module and a probe module. The dependency module is built first. The dependency module ensures that the library dependencies are correctly installed to the Maven repository before the probe module is built.

Deploy Probe

You can deploy a probe through Admin Console or Infrastructure Manager.

Deploy in Admin Console

Follow these steps:

1. Click the Archive button at the top of the Admin Console window.
2. Click Local Archive.
3. Select the Import button and find the <probe_name>.zip file.
4. Click **Ok**.

   You receive a confirmation message.

   The probe package is downloaded to your local archive.

5. Deploy the probe by clicking the robot in the left navigation panel, selecting the probe, and then pressing the Deploy button.

Deploy in Infrastructure Manager

Follow these steps:

1. Drag the probe package <probe_name>.zip from the probe directory into the Infrastructure Manager archive.

2. Find the probe in the package archive window, and then drop the probe onto a domain node, hub node, or robot node under Domains.

Chapter 5: Create a Probe

This section describes how to modify probe-specific files and create a new probe. This document does not cover standard Java development principles. The probe development process focuses on the modification of a few key files that are described in the following sections.

Note: If you have previously used the CA Nimsoft Java Developer's Guide, be aware that that document does not address all the requirements that you need for the current probe architecture model.

Files for Probe Development

The following sections describe the files that you must review and modify to create the probe architecture.

Probe_schema.cfg

The probe_schema.cfg file is required for any metric group or topology probe. This file defines the element types and their associated QoS metric types. The element types model the probe inventory which the probe publishes as the discovered topology. The metric types define the QoS data that can be published for the element type.

The probe_schema.cfg file contains a section to define an element, CTD type, and metric types. In the element types section, you specify the CTD type. The CTD type is one of the standard set of CTD classes with properties customized for the type. See the MetricTypesWithUnits.SpreadSheetOfMostCommonTypesForReuse file in the docs directory in probe-sdk-*<version>*.zip for recommended CTD types for various metrics. If your custom monitor type in your probe_schema.cfg file uses metrics from various CTD types, pick the metric that best fits your monitor type.

Element Types

This section of the file defines the element types and element properties. Each element type contains the associated metric types. You can define additional properties to extend the base element for each element type. For example, you can specify an icon for an element type.

Example element type section with properties:

```
<probe_schema>
    <element_types>
        …
        …
```

```
…
<StorageFile>
      <properties>
          base_element_type = FileElement
          icon = memory
      </properties>
      <qos_metric_types>
          <FileName>
              active = yes
              descr = Name of the file
              unit = String
              name_label = FileName
              metric_type = 1:14
              qos_name =
          </FileName>
…
…
…
```

QoS Metric Types

This section of the file contains the metric definitions. The possible parameters are:

- Active - Indicates if the section is processed as a metric. Enter yes to process the section as a metric, and enter no to avoid processing the section as a metric.
- Unit – The units for the metric. These units are defined in in CI manager and are automatically added to the file if you use a standard CI type ID.
- Description - Descriptive information for the metric.
- Name Label – The display name for the metric.
- Metric Type - The numeric string that represents a CI type and the associated MI type.
- QoS Name - The QoS metric name.

  Important! The name of any custom QoS metrics you add must follow the format of QOS_<APPLICATION/PROBE_NAME>_<UNIQUE_IDENTIFIER> by convention. The entire metric name is capitalized with no spaces by convention. For example, QOS_CLOUD1_MONITORNAME. It is also a good practice to append the abbreviated units.

The following section shows some example of a QoS measurement in the StorageDirectory element:

```
…
…
…

   <StorageFile>
         <properties>
             base_element_type = FileElement
             icon = memory
         </properties>
         <qos_metric_types>
             <FileName>
                 active = yes
                 descr = Name of the file
                 unit = String
                 name_label = FileName
                 metric_type = 1:14
                 qos_name =
             </FileName>
…
…
…
```

pom.xml

A Maven project file is provided for your convenience if you prefer to use Maven for your build automation. When you use Maven, you can also specify any extra library dependencies in the pom.xml file.

The probe project is a modular project. The top-level pom.xml file in the working probe directory shows the modules to be built. This file has version numbers set for dependencies that are referenced by the lower level project pom.xml files.

The sub-projects are in the dependencies, probe, and .zip directories. The dependencies project is built first, and adds dependency libraries to the local Maven repository. The probe project is built next, and creates the <probe_name>.jar file. The .zip project is run last, and creates the probe .zip archive that can be loaded through Admin Console or Infrastructure Manager.

<probe>.cfx

The <probe>.cfx file defines any default setup and startup information, resources, and alarm messages for the probe. This file also functions as a template for the <probe>.cfg file that the Probe-

SDK automatically generates. The following sections describe the contents that are typically found in the file.

Setup

This section of the file contains the general parameters for the probe. These parameters include:

- Loglevel - The amount of information that is generated by the running probe. The default level for a probe is 3.
- Logsize - The maximum size of the log file in MBs.

Example section:

```
<setup>
   loglevel = 1
   logsize = 3000
</setup>
```

Startup

This section of the file contains startup options for Xms memory settings and the default probe language.

Example section:

```
<startup>
   options = -Xms32m -Xmx512m -Duser.language=en -Duser.country=US
</startup>
```

Messages

All possible messages are available in the <probe>.cfx file by default. Many of the default messages might not make sense for your probe. It might be necessary to view this file and remove the messages you do not want to use with the probe.

There is typically one section for each message which includes the following properties:

- Name - A descriptive name for the message.
- Token – A classification for filtering lists of alarms.
- Message text strings – The text strings for the alarm message and the clear alarm message.
- Level – The alarm severity
- Subsystem ID – The NAS subsystem identification number sequence, separated by dots, identifying the subsystem. Use this ID to categorize the alarms sent along with the alarm message from the probe. You can look up the available subsystem IDs through the nas probe interface. For more information, see (Optional) Update NAS Subsystem IDs.

For example, the subsystem ID 2.7.1 is a subsystem ID for a VMware resource.

2.= The group for software vendors

2.7. = VMware

2.7.1=Resource

Example section:

```
<messages>
   <ResourceCritical>
      name = ResourceCritical
      token = resource_error
      msn_err = $host is not responding (reason: $descr)
      msn_ok = $host is now responding
      sev = 5
      subsystem = 2.7.1
   </ResourceCritical>
</messages>
```

<probe>.pkg

The <probe>.pkg file defines the package information of the <probe>.zip file. This file points to the GUI executables, auto-generated localization files, and required files for the probe. You can deploy files generically across platforms, and you can deploy specific files on specific platforms. The following sections describe the contents that are typically found in the file.

Package Info

This section of the file contains general information about the probe package.

Example section:

```
<package info>
   name = cloud1
   description = Basic Devkit Probe
   copyright = Copyright 2014, CA. All rights reserved.
   group = Application
   author = MyCompany
   version = 1.00
   build = 01
</package info>
```

Pre_install

This section of the file contains package information about packages that are required to run before probe deployment. Do not change the information in this section.

Example section:

```
<pre_install>
   update = cloud1
   post-install = $NIM_JRE_HOME/bin/java -jar
probes/custom/cloud1/pre_install/packageCleaner.jar
probes/custom/cloud1/pre_install/pre_install.xml
   <files>
      <packageCleaner.jar>
         type = binary
         access = 0644
         dir = probes/custom/cloud1/pre_install
      </packageCleaner.jar>
      <pre_install.xml>
         type = text
         access = 0644
         dir = probes/custom/cloud1/pre_install
      </pre_install.xml>
    </files>
</pre_install>
```

Generic

This section of the file defines the deployment of the <probe>.cfx, <probe>.cfg, and <probe>.jar files. The deployment definition includes:

- Platform dependencies
  To deploy specific files on a specific platform you must add a section to the <probe>.pkg file for each platform. You must define the platform-specific files and dependencies within each platform section.
- Default directory and log file locations
- Required library files

The following example shows the generic section where the cloud1.cfx, probe_schema.cfg, and cloud1.jar files are deployed on any platform.

```
<generic>
   name = cloud1
   type = cloud1
   update = cloud1
   <cloud1>
      description = Cloud1 Probe
      group = Application
```

```
      active = yes
      preserve_state = yes
      type = daemon
      timespec =
      workdir = probes/custom/cloud1
      command = <startup java>
      arguments = -jar cloud1.jar
      config = cloud1.cfg
      logfile = cloud1.log
      datafile =
      security = write
   </cloud1>
   <files>
      <cloud1.jar>
         type = binary
         access = 0644
         dir = probes/custom/cloud1/lib
      </cloud1.jar>
      <cloud1.cfx>
         type = config
         access = 0644
         dir = probes/custom/cloud1
      </cloud1.cfx>
      <probedef.cfg>
         type = config
         access = 0644
         dir = probes/custom/cloud1
      </probe_schema.cfg>
   </files>
</generic>
```

The following example shows the generic section where the Linux-specific section within a <probe>.pkg file. The cloud1.jar, cloud1.cfg, and cloud1.log files are installed only on a Linux platform with a robot version of 4.10 and a Java jre version of 1.6.0.

```
<LINUX>
   name = cloud1
   OStype = unix
   OS = LINUX
   update = cloud1
   <cloud1>
      description = Cloud1 Probe
      group = Custom
      active = yes
      preserve_state = yes
      type = daemon
      timespec =
```

```
      workdir = probes/custom/cloud1
      command = <startup java>
      arguments = -jar cloud1.jar
      config = cloud1.cfg
      logfile = cloud1.log
      datafile =
      security = write
   </cloud1>
   <dependencies>
      <Robot>
         version = 5.23
         type = ge
      </Robot>
      <java_jre>
         version = 1.6.0
         type = ge
      </java_jre>
   </dependencies>
   <files>
   </files>
</LINUX>
```

Note: For a list of supported platforms, see the CA UIM Compatibility Matrix.

Add Libraries and Other Dependencies

When you use Maven for your builds, you can specify additional dependencies in the pom.xml file included in the copy of your probe project. For example, you can add more libraries. See the Maven documentation for information about adding dependencies.

(Optional) Update NAS Subsystem IDs

This task is only required if you need to configure custom alarms and messages. Alarms are classified by their subsystem ID, identifying which part of the system the alarm relates to. These subsystem IDs are kept in a table maintained by the NAS probe. You can add subsystem IDs manually to CA UIM using the NAS Raw Configuration menu. For more information about the nas probe, see the nas Guide.

Update the Subsystem IDs in Admin Console

Follow these steps:

1. In the Admin Console, click the black arrow next to the NAS probe, select **Raw Configure**.
2. Click on the **Subsystems** folder.
3. Locate the appropriate section. For example, Private.
4. Click on the New Key Menu item.
5. Enter the Key Name in the Add key window, click **Add.**
   The new key appears in the list of keys with a blank value.
6. Click in the Value column for the newly created key and enter the key value.
7. Repeat this process for all of the required subsystem IDs for your probe.
8. Click **Apply**.

Update the Subsystem IDs in Infrastructure Manager

Follow these steps:

1. In Infrastructure Manager, right click on the NAS probe, select **Raw Configure**.
2. Click on the **Subsystems** folder.
3. Locate the appropriate section. For example, Private.
4. Click on the **New Key...** button.
5. Enter the Key Name and Value, click **OK.**
6. Repeat this process for all of the required subsystem IDs for your probe.
7. Click **Apply**.

Debugging a Probe

An IDE is not required to create a probe, but an IDE contains useful utilities that you can use to view, modify, or debug your code. The following section describes how to attach the JPDA (Java Platform Debugger Architecture) to an IDE.

1. Load the main pom.xml file into an IDE. Open the <probe>.cfg file and go to the startup section. You can view the startup section in a text editor or through raw configure.
2. Append the following string to the options value:

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,
address=<port to use>
```

Example:

```
options = -Xmx64m -Xmx1024m -
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,
address=1044
```

3. Attach the debugger to your IDE. For more information, see the instructions that are provided for your particular IDE.

   For example, the install directory is
   C:\Program Files (x86)\Nimsoft\probes\application\local_dirscan1,
   and the development directory is
   C:\workspace\stage_testing\ProbeFramework\devkit\examples\local_dirscan1\probe.

Chapter 6: Example

This section uses examples to take you through the probe development process.

Copy and Paste Mikes Doc