CA Unified Infrastructure Management

Remote Probe with Bulk Configuration Example

V1.0

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 © 2015 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

Advanced Probe Development 5

Get the Example Source Code 5
Example Probe Overview 6
    Example Probe Demo 6

Inventory Modeling Example 11

Define Model and Data Use 11
    Topology Overview 11
    Available Inventory Modeling Classes 11
    Topology Entity Breakdown 12
    Source, Target, and Naming Breakdown 15
Declare Inventory Elements and Metrics in probe_schema.cfg 15
    Generate Inventory Elements 16
Differences between Topology Structure and UI Presentation 16
Implement Inventory and Metric Publishing 17
    Implement IProbeInventoryCollection.getUpdatedInventory() 18

Bulk Configuration Example 21

Probe Requirements to Support Bulk Configuration 21
Process Overview 21
Implementing your TemplateDefinitionCreator 22
    Extend AbstractTemplateDefinitionCreator 22
    Define Basic Inventory Structure with GenMonitorTypes 23
    Implement Map Getters to Allow Framework Access 24
    Implement createEntityInfos() 25

Advanced Probe Development

This article contains the more advanced probe development topics of inventory modeling and bulk configuration.

Important! You should have a basic understanding of the probe development process with the Probe SDK and understand how to use the example probes provided with the SDK before proceeding.

In this article we investigate the probe development topics of inventory modeling and bulk configuration. We will review the elements that make up the "Mock VM Host" probe. This probe is provided with the Probe SDK. To get the most out of this article, you should deploy an instance of this probe to your CA UIM environment and open the source code in your favorite IDE. This will allow you to look at the probe with this article as a guide.

Get the Example Source Code

If you have installed the Probe SDK, you can use the provided Maven Archetype to generate a probe project based on the "Mock VM Host" probe. For more information about Maven Archetypes, see https://maven.apache.org/guides/introduction/introduction-to-archetypes.html.

The code for this example is provided in the Archetype titled: **com.nimsoft:mock_vm_host**. You can create your own probe project from this template by opening a command prompt in the location you wish to create your project and issue the command: `mvn archetype:generate -Dfilter=com.nimsoft:devkit-mock_vm_host-probe` This will kick off the Maven archetype generation process in interactive mode. Below is a capture of what the interaction will look like:

Project Generation with Maven Archetype

```
mvn archetype:generate -Dfilter=com.nimsoft:mock_vm_host
1: local -> com.nimsoft:mock_vm_host
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive
contains): :
Define value for property 'groupId': : org.example
Define value for property 'artifactId': : MyMockVMProbe
Define value for property 'version':  1.0-SNAPSHOT: : 0.5
Define value for property 'package':  org.example: : org.example
Confirm properties configuration:
groupId: org.example
artifactId: MyMockVMProbe
version: 0.5
package: org.example
 Y: : y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: devkit-
mock_vm_host-probe:2.1.0-

SNAPSHOT
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: org.example
[INFO] Parameter: artifactId, Value: MyMockVMProbe
[INFO] Parameter: version, Value: 0.5
[INFO] Parameter: package, Value: org.example
```

```
[INFO] Parameter: packageInPathFormat, Value: org/example
[INFO] Parameter: package, Value: org.example
[INFO] Parameter: version, Value: 0.5
[INFO] Parameter: groupId, Value: org.example
[INFO] Parameter: artifactId, Value: MyMockVMProbe
[INFO] project created from Archetype in dir: C:\tmp\AA\MyMockVMProbe
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

Example Probe Overview

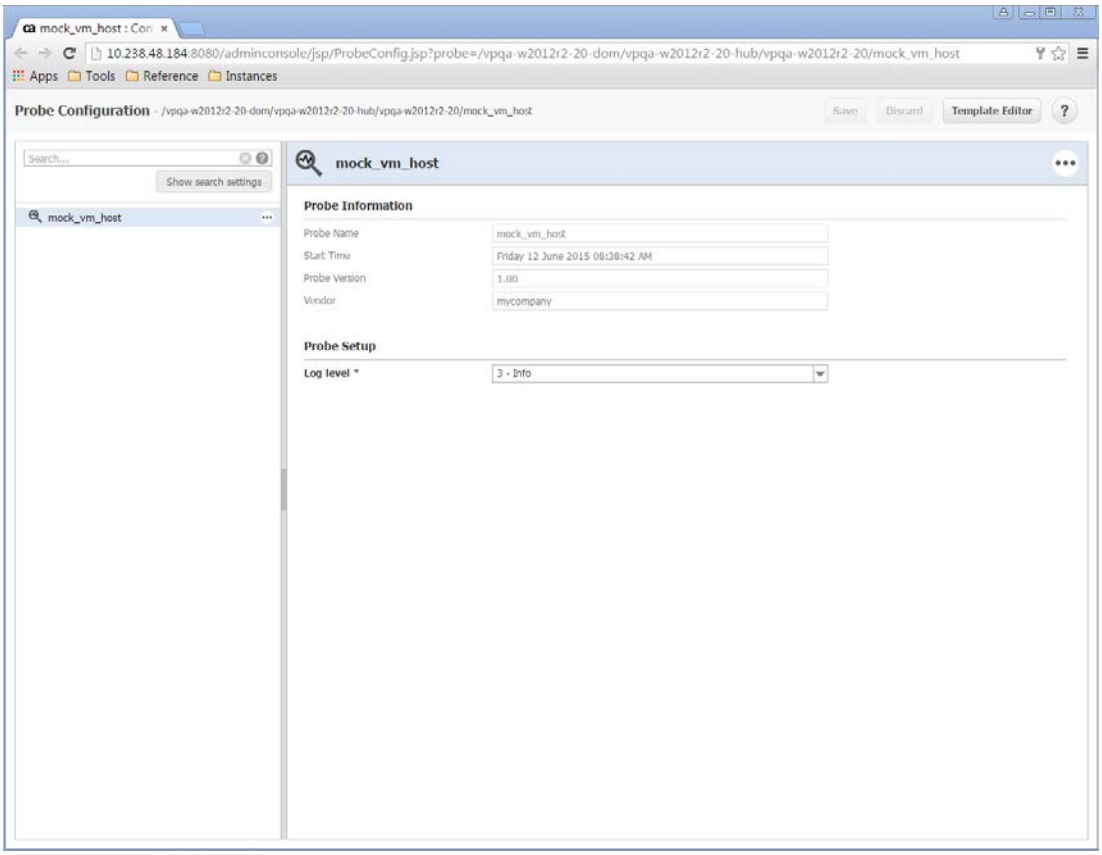
The "Mock VM Host" probe is an example probe provided in the Probe SDK. This probe simulates monitoring a VM hosting environment. This probe demonstrates a typical remote probe where the system being monitored is external to the environment in which the probe is running. The probe publishes an inventory that represents VM servers as well as the individual VM instances running on each host. You do not need an actual VM environment to monitor since this probe includes a data collection service that simulates the external communication and provides all mock data.

The probe development concepts that can be learned from studying this example are:

1. The probe lifecycle: Startup, registration, configuration, polling cycle.
2. Inventory modeling
3. Inventory and metric publishing
4. Bulk configuration

Example Probe Demo

To see the probe in action, you must deploy it on a robot and open the probe configuration GUI. The following diagram shows an example of the probe configuration GUI:



Configure a New Profile or Resource

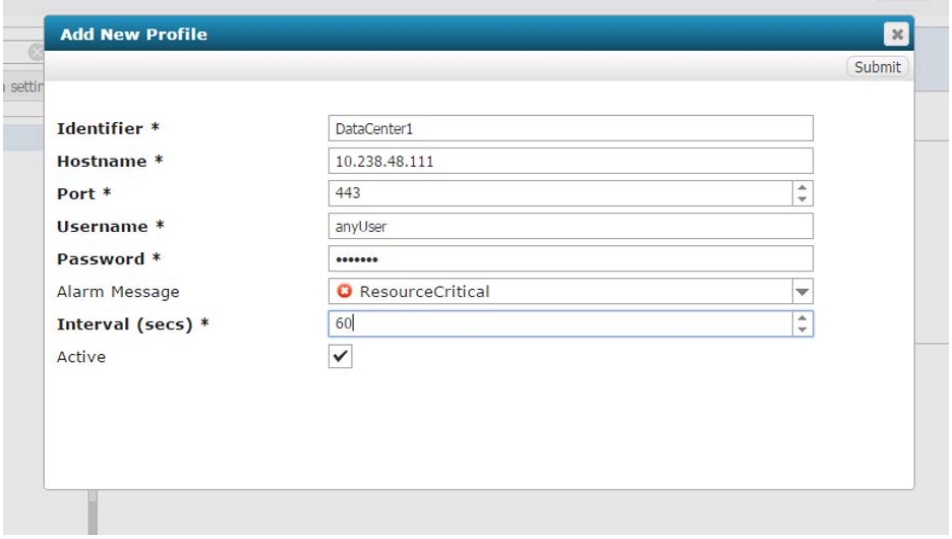
To configure a new profile or resource:

1. Go to the probe configuration GUI.
2. Select the Options (**...**) icon, and then select **Add New Profile**.
3. Complete the fields in the Add New Profile window.
   Fields to know:

Name	Description
Identifier	Profile name
Hostname	Fully qualified hostname or IP address of the remote system. This is a standard property that all remote probes must have. You normally must provide this information for a probe, but in this example the information is not provided. This is because this probe example is only simulating the remote communication.

Port	Port we will use to communicate with the remote system. This is a standard property that all remote probes must have. You normally must provide this information for a probe, but in this example the information is not provided. This is because this probe example is only simulating the remote communication.
Username	Username for communicating with remote system. This is a standard property that all remote probes must have. You normally must provide this information for a probe, but in this example the information is not provided. This is because this probe example is only simulating the remote communication.
Password	Password for communicating with remote system. This is a standard property that all remote probes must have. You normally must provide this information for a probe, but in this example the information is not provided. This is because this probe example is only simulating the remote communication.
Alarm Message	The alarm that should be published if communication with the resource is lost.
Interval	Polling interval. The frequency at which the Probe SDK will initiate a data collection and publishing cycle.
Active	Is resource active

The following figure shows an example of the add New Profile window.

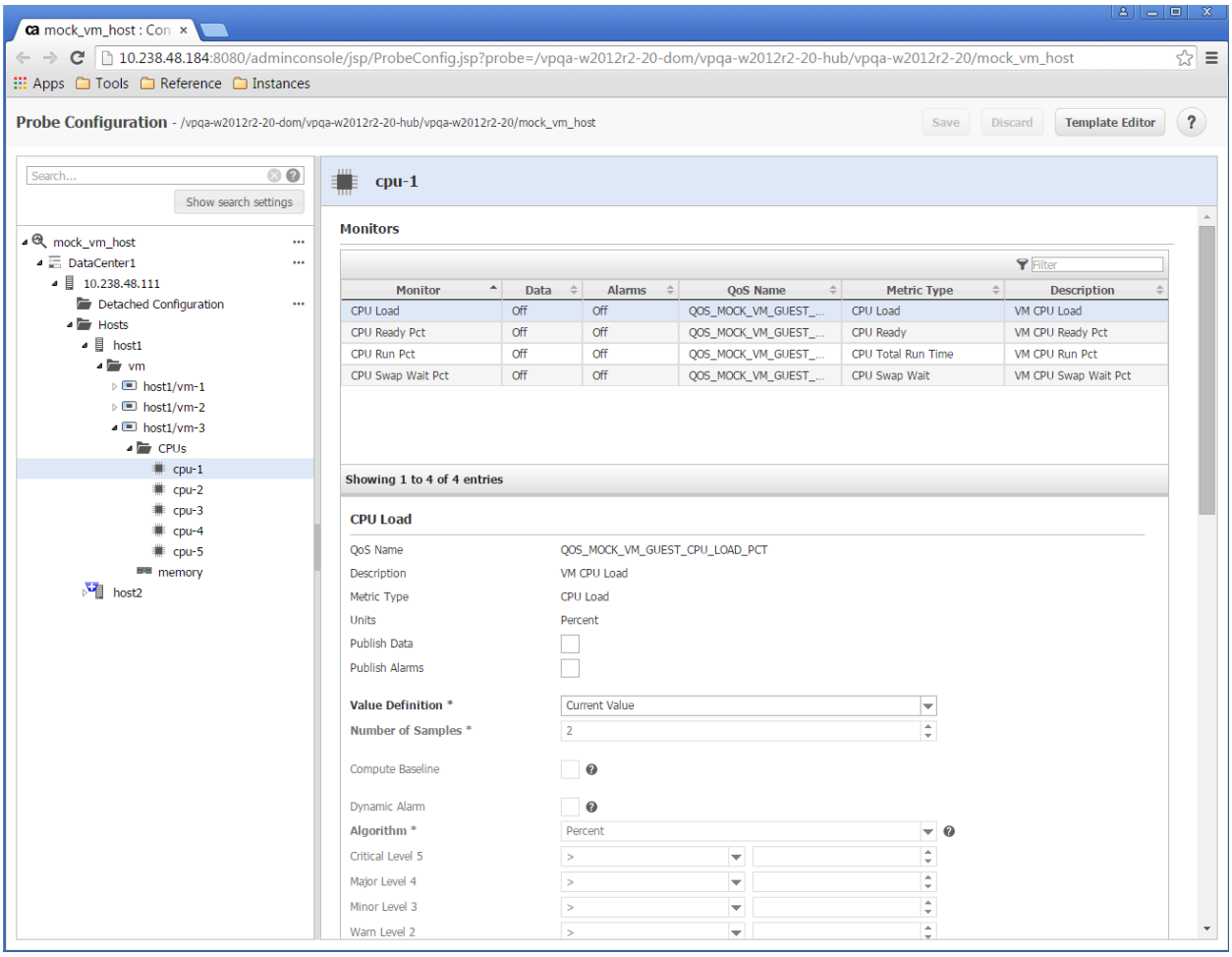


Viewing the Inventory

After you save the new profile and refresh the GUI you will see an inventory of the profile elements. The inventory is made up of:

- The Resource (or Profile) which we called 'Datacenter1' and specified by IP address.
    - A folder called 'Hosts'. Organizing inventory into folders is a topic that we will discuss in the inventory modeling section of this document.
        - Multiple 'Host' elements. Each host represents a VM server in our inventory. We will associate metrics here.
            - A folder called 'vm' that contains all of the VM instances running on that particular host.
                - Multiple 'VM' inventory elements, each with associated metrics.
                - A folder called 'CPU's'
                - Multiple 'CPU' inventory elements, each with associated metrics.
                - A single 'memory' inventory element, with associated metrics.

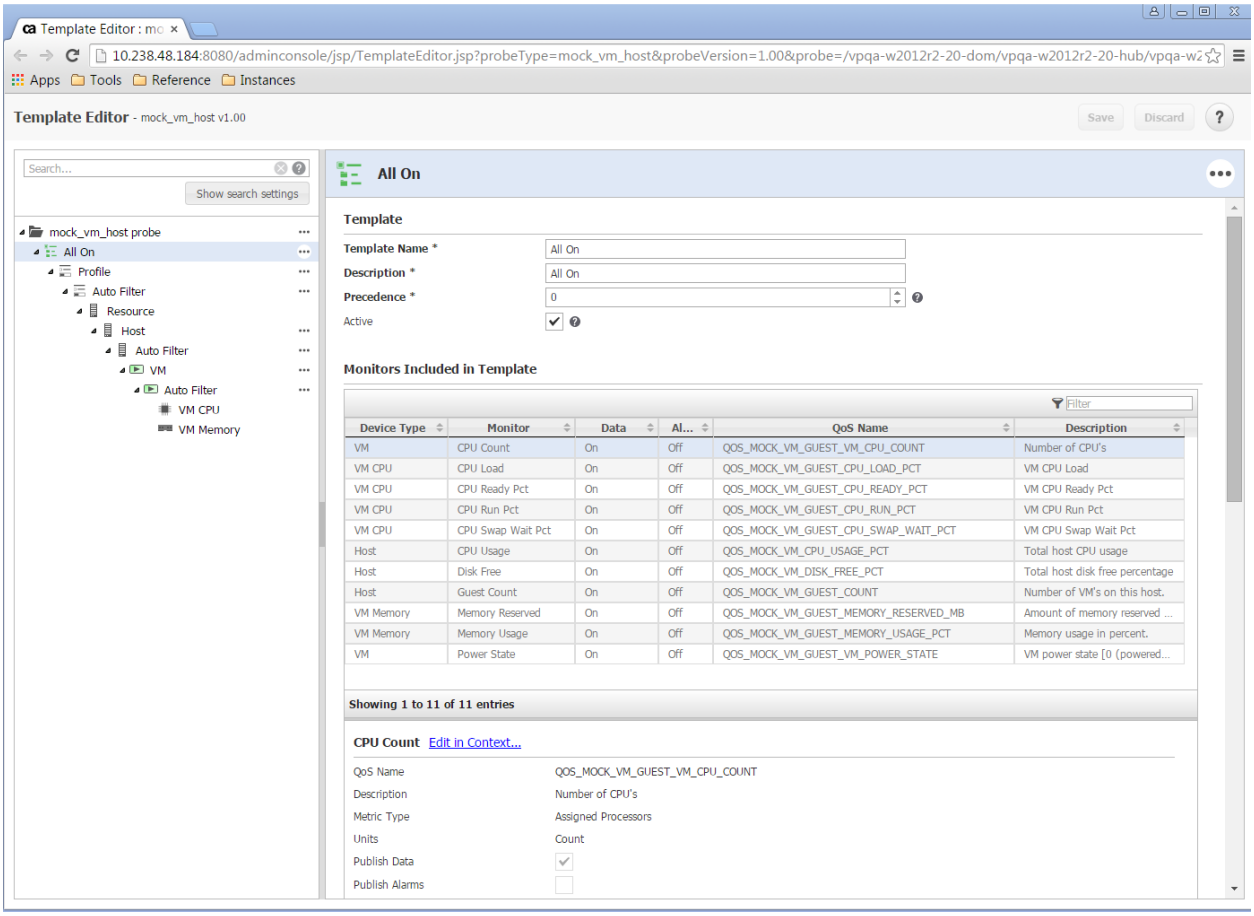
The following figure shows an example of the probe inventory.



Template Editor

The Mock VM Host probe also demonstrates the bulk configuration functionality by including the Template Editor. You open the Template Editor by clicking the button in the upper right portion of the probe configuration GUI. The Template Editor opens in a new tab. The Template Editor allows you to define a new template for applying measurements and thresholds to multiple profiles at once.

The following figure shows a new template called "All On".



Inventory Modeling Example

This section describes the inventory modeling process from design to implementation.

Define Model and Data Use

Before you write a single line of code you need to understand the following items:

1. What data will my probe collect?
2. What measurement and inventory data will be sent to UIM?
3. For each measurement, what will be my source, target, CI type and QOS name?
4. How will the data from this probe be used? Reports? custom dashboards?
5. What will my inventory graph look like?

Topology Overview

This is the design information that was created while designing the "Mock VM Host" probe. This probe is in the Probe SDK as a Maven Archetype.

Topology Tree View

```
RESOURCE
        Hosts(folder)
                MyHost
                        vm(folder)
                                MyVM
                                        CPUs(folder)
                                                MyVMCPU
                                        MyVMMemory
```

Available Inventory Modeling Classes

In this example we are modeling inventory elements that are called MyHost, MyVM, MyVMCPU, and MyVMMemory. Additional details must be provided for each inventory element. First decide what Probe SDK API classes to use to model the inventory elements. To see the available classes, refer to the API JavaDoc for the com.nimsoft.probe.framework.devkit.inventory package. The JavaDoc can be found in the Probe SDK.

The following figure shows the JavaDoc page:

Overview Package Class Use Tree Deprecated Index Help

Prev Package Next Package Frames No Frames

Package com.nimsoft.probe.framework.devkit.inventory

This package contains all the classes the Probe Framework provides for inventory modeling and Metric reporting.

See: Description

Class Summary

Class	Description
ClusterElement	Inventory modeling class that represents a group of compute resources in a virtual environment.
CollectionElement	Inventory modeling class for other metric groupings and aggregates.
ComputerSystemElement	Inventory modeling class that represents a `ComputerSystem Element`
DiskPartitionElement	Inventory modeling class that represents a `DiskPartition Element`
Element	Element(s) are used to model a probe's inventory of components with metrics ...
Element.ChildParentRelationship	
Element.ElementCounter	
Element.PropertyDefinitions	
Element.TypeDefinition	
FileElement	Inventory modeling class that represents a `File Element`
Folder	Folders are a pure UI construct and are only for organizational purposes.
FolderView	Deprecated
HardwareElement	Inventory modeling class that represents a `RunningHardware Element`
HypervisorManagerElement	Inventory modeling class that represents a `HypervisorManager Element`
IPDeviceElement	Inventory modeling class that represents a `IPDevice Element`
MemoryElement	Inventory modeling class that represents a `Memory Element`
NetworkInterfaceElement	Inventory modeling class that represents a `NetworkInterface Element`
PortElement	Inventory modeling class that represents a `Port Element`
ProcessorElement	Inventory modeling class that represents a `Processor Element`
RouterElement	Inventory modeling class that represents a `Router Element`
SoftwareElement	Inventory modeling class that represents a `RunningSoftware Element`
StorageVolumeElement	Inventory modeling class that represents a `StorageVolume Element`
SwitchElement	Inventory modeling class that represents a `Switch Element`
VirtualSystemElement	Inventory modeling class that represents a `VirtualSystem Element`

Package com.nimsoft.probe.framework.devkit.inventory Description

This package contains all the classes the Probe Framework provides for inventory modeling and Metric reporting. All of the classes in this package that end in 'Element' are valid to use for the element-types you declare in probe_schema.cfg.

When you model your inventory element after one of the Element's contained here (in probe_schema.cfg) you will also attach the Metric Definitions you wish to report.

Note: This package also contains `Folder` which is a pure UI construct and are only for organizational purposes. Folders should not include metrics.

Topology Entity Breakdown

After selecting the appropriate API classes to model the inventory elements, you can produce the following tables with the entity design details.

MyHost

ID	hostname or IP
Element	com.nimsoft.probe.framework.devkit.inventory.ComputerSystemElement
Parent	hosts folder
Children	• VM's folder

Metrics	• QOS_MOCK_VM_GUEST_COUNT
	  o Number of VM's on this host
	  o 1.16:11
	• QOS_MOCK_VM_CPU_USAGE_PCT
	  o Total host CPU usage
	  o 1.5:1
	• QOS_MOCK_VM_DISK_FREE_PCT
	  o Total host disk free percentage
	  o 1.1:14

MyVM

ID	<parent hostname>/<vm name or ip>
Element	com.nimsoft.probe.framework.devkit.inventory.VirtualSystemElement
Parent	VM's folder
Children	• CPU's folder • MyVMMemory Element

Metrics	• QOS_MOCK_VM_GUEST_VM_POWER_STATE
	  o VM power state
	  o 1.17:1
	• QOS_MOCK_VM_GUEST_VM_CPU_COUNT
	  o Number of CPU's
	  o 1.17.1:5

MyVMCPU

ID <parent VM>:cpu ID>

Element com.nimsoft.probe.framework.devkit.inventory.ProcessorElement

Parent CPU's folder

Children None

Metrics

- QOS_MOCK_VM_GUEST_CPU_LOAD_PCT
  - VM CPU Load
  - 1.17.1:16
- QOS_MOCK_VM_GUEST_CPU_RUN_PCT
  - VM CPU Run Pct
  - 1.17.1:15
- QOS_MOCK_VM_GUEST_CPU_READY_PCT
  - VM CPU Ready Pct
  - 1.17.1:21
- QOS_MOCK_VM_GUEST_CPU_SWAP_WAIT_PCT
  - VM CPU Swap Wait Pct
  - 1.17.1:26

MyVMMemory

ID <parent VM>: memory

Element com.nimsoft.probe.framework.devkit.inventory.MemoryElement

Parent MyVM

Children None

Metrics

- QOS_MOCK_VM_GUEST_MEMORY_USAGE_PCT
  - Memory usage in percent
  - 1.17.2:19
- QOS_MOCK_VM_GUEST_MEMORY_RESERVED_MB
  - Amount of memory reserved in MB
  - 1.17.2:3

Source, Target, and Naming Breakdown

As part of the design work you should think about what will be the source, target, and name for the metrics associated with each element. To do this properly, you need to understand how the data will be used and how to identify the data for reporting.

MyHost

```
Name:               <hostname>
Source:    MyHost
```

MyVM

```
Name:               <hostname>/<vm name>
Source:    MyVM
```

MyVMCPU

```
Name:               <vm name>:cpu id
Source:    MyVM
```

MyVMMemory

```
Name:               <vm name>:memory
Source:    MyVM
```

Declare Inventory Elements and Metrics in probe_schema.cfg

The file probe_shema.cfg is where you declare your inventory elements and associated metrics. If you have done your design work and have determined how your inventory and metrics will be modeled then the configuration of this file is straight forward.

The following is an excerpt from the file showing the definition for the 'MyHost' element and its associated metrics.

probe_schema.cfg

```
<probe_schema>
    <element_types>
                    ...
        <MyHost>
            <properties>
                base_element_type = ComputerSystemElement
            </properties>
            <qos_metric_types>
                <GuestCount>
                    unit = Count
                    active = yes
                    descr = Number of VM's on this host.
                    name_label = Guest Count
                    metric_type = 1.16:11
                    qos_name = QOS_MOCK_VM_GUEST_COUNT
                </GuestCount>
                <CPUUsage>
                    unit = Percent
                    active = yes
```

```
                    descr = Total host CPU usage
                    name_label = CPU Usage
                    metric_type = 1.5:1
                    qos_name = QOS_MOCK_VM_CPU_USAGE_PCT
                </CPUUsage>
                <DiskFree>
                    unit = Percent
                    active = yes
                    descr = Total host disk free percentage
                    name_label = Disk Free
                    metric_type = 1.1:14
                    qos_name = QOS_MOCK_VM_DISK_FREE_PCT
                </DiskFree>
            </qos_metric_types>
        </MyHost>
           ...
    </element_types>
</probe_schema>
```

Some things to consider in the probe_schema.cfg example are:

1. Line 6 specifies that the MyHost element is based on the 'ComputerSystemElement' type. Recall that this is one of the inventory modeling classes available in the com.nimsoft.probe.framework.devkit.inventory package.
2. Lines 8-33 declare the metrics that are attached to this element. Specify the QoS name, metric type, and units that were determined in the probe design work.

Generate Inventory Elements

After configuring probe_schema.cfg you must run the command `mvn generate-sources` to auto generate Java classes that represent your Inventory Elements.

Note: This will also occur automatically every time you do a complete build and package of your probe using the `mvn install` command. This ensures that the generated classes always accurately reflect the contents of probe_schema.cfg.

Differences between Topology Structure and UI Presentation

To properly build a topology, you must understand the conceptual difference between the logical topology versus the GUI representation. This is because you often add additional GUI elements for display purposes only. For example, for the Mock VM Host probe the GUI view of the topology looks like this:

GUI Representation of Topology

```
RESOURCE
        Hosts(folder)
                MyHost
                        vm(folder)
                                MyVM
                                        CPUs(folder)
                                                MyVMCPU
                                        MyVMMemory
```

But when the probe publishes this topology to the bus you do not want to represent the Folders, since they are for GUI purposes only. So the published topology looks like this:

Actual Topology

```
RESOURCE
        MyHost
                MyVM
                        MyVMCPU
                        MyVMMemory
```

You must be mindful of this as you implement the code that will build the topology. When you create new topology elements, you have the ability to specify both the "Topology Parent" and the "GUI Parent". Consider this following code snippet:

Topology Parent vs. GUI Parent

```
// construct a host Element that is a child of ResourceConfig, but for UI purposes
will be in a Folder
Element topologyParent = resourceConfig;
Element uiParent = hostsFolder;
MyHost host = MyHost.addInstance(inventoryDataset, new EntityId(topologyParent,
hostname), hostname, uiParent);
```

This example shows how you can have a different topology parent vs. a GUI parent.

Implement Inventory and Metric Publishing

If your probe produces inventory and metrics, your probe must implement the *IProbeInventoryCollection* interface. The method *IProbeInventoryCollection.getUpdatedInventory()* is invoked by the framework on the polling interval. This method gives the probe an opportunity to return the latest inventory and metric data.

Please see the following pseudo code that describes this process.

1. Construct a new InventoryDataset
2. For each item in your inventory
    a. Create a new instance of the inventory element using the static factory 'addInstance()' method. You pass a reference of the InventoryDataset as well as the parent object. For the top most item or items in your inventory structure the parent will be the ResourceConfig object.
    b. For each metric you wish to set on this inventory element
        i. Call the 'setMetric()' method on the element, passing the metric name and the value.
3. Return the newly created InventoryDataset that now contains a collection of all inventory items and metrics, as well as the ResourceConfig which contains the parent/child structure information.

Implement IProbeInventoryCollection.getUpdatedInventory()

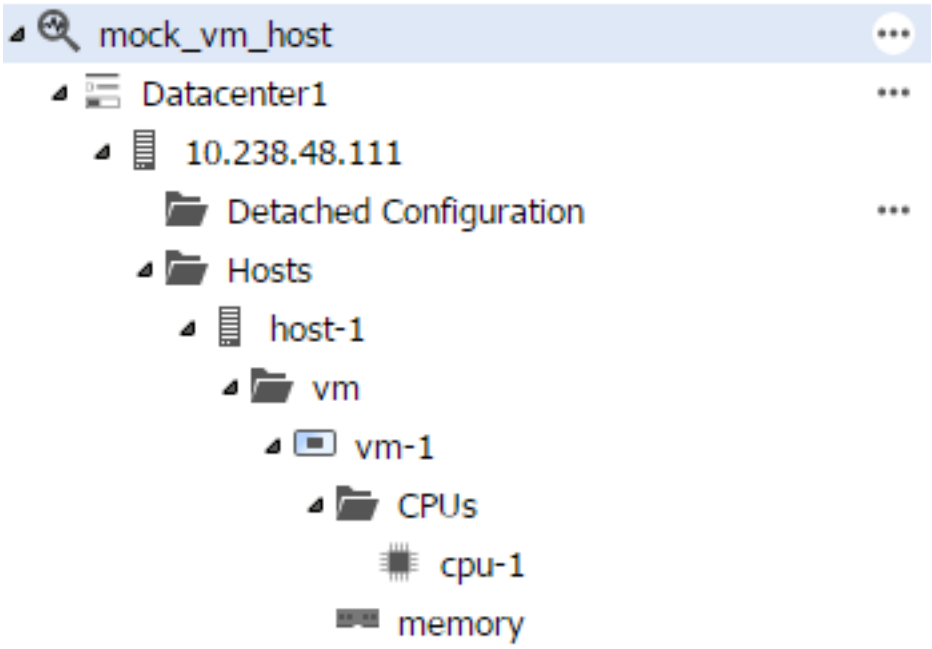
Build Element Topology

For example, look at a simple implementation of getUpdatedInventory() which produces a hard coded inventory. The following example has inventory elements called MyHost, MyVM, MyVMCPU, and MyVMMemory as well as a couple of folders for grouping.

Inventory and Metrics

```
RESOURCE
        Hosts(folder)
                MyHost
                        vm(folder)
                                MyVM
                                        CPUs(folder)
                                                MyVMCPU
                                        MyVMMemory
```

In this example, you want to publish a very simple inventory. You want the inventory to look exactly like the inventory in the following figure.



You can accomplish this with the implementation of getUpdatedInventory() shown in the following example.

Note: This example is overly simplified as it is completely hard coded with no attached metrics. The purpose of this example is to demonstrate how to construct a topology.

Simple getUpdatedInventory()

```
1  public IInventoryDataset getUpdatedInventory(ResourceConfig resourceConfig,
       IInventoryDataset previousDataset) throws NimException, InterruptedException {

2    InventoryDataset inventoryDataset = new InventoryDataset(resourceConfig);

3    Folder hostsFolder = Folder.addInstance(inventoryDataset,
         new EntityId(resourceConfig,"hosts"), "Hosts", resourceConfig);

4    MyHost host1 = MyHost.addInstance(inventoryDataset,
         new EntityId(resourceConfig, "host-1"), "host-1", hostsFolder);

5    Folder vmsFolder = Folder.addInstance(inventoryDataset,
         new EntityId(host1,"vm"), "vm", host1);

6    MyVM myVM1 = MyVM.addInstance(inventoryDataset,
         new EntityId(host1, "vm-1"), "vm-1", vmsFolder);

7    MyVMMemory myMemory = MyVMMemory.addInstance(inventoryDataset,
         new EntityId(myVM1, "memory"), "memory", myVM1);

8    Folder vmCPUFolder = Folder.addInstance(inventoryDataset,
         new EntityId(myVM1,"CPUs"), "CPUs", myVM1);

9    MyVMCPU myCPUElement = MyVMCPU.addInstance(inventoryDataset,
         new EntityId(myVM1, "cpu-1"), "cpu-1", vmCPUFolder);

10   return inventoryDataset;

11 }
```

Some things to consider in this example are:

- Line 3: Create a new, empty InventoryDataset. Pass this to the *.addInstance() method of every element we create.
- Speaking of the addInstance() method, notice that it takes 4 parameters which are (InventoryDataset, EntityId, label, and guiParent), As you look at the code, notice that the 4th parameter dictates how the elements are related in the GUI.
- Line 5: Create the "Hosts" folder as a child of the ResourceConfig.
- Line 6: Add "host-1" element to the Hosts folder.
- Line 7: Create a "vm" folder under host-1
- Line 8: Add "vm-1" element to the vm folder
- Line 9: Add a memory element to vm-1
- Line 10: Add a "CPUs" folder under vm-1
- Line 11: Add a "cpu-1" element to the CPUs folder

Set Metrics on Element

The previous example demonstrated how to create inventory elements and place them in parent/child relationships to model a topology, but did not demonstrate how to set metrics on those elements. The following example demonstrates how metrics are set on elements.

Setting Metrics

```
                ...
                MyHost host1 = MyHost.addInstance(inventoryDataset, new
EntityId(hostsFolder, "host-1"), "host-1", hostsFolder);
                host.setMetric(MyHost.GuestCount, vmGuestsJsonArray.size());
        host.setMetric(MyHost.DiskFree, diskFreePercent);
        host.setMetric(MyHost.CPUUsage, cpuUsagePercent);
                ...
```

In this example, you call the setMetric() method and pass in the Metric Definition and the value.

Set Additional Properties on Elements

The Probe SDK give you the ability to set additional properties on your elements. The available properties are dependent on which of the classes in the `com.nimsoft.probe.framework.devkit.inventory` package you used to model your Element. Recall that the "MyHost" element was modeled as a `ComputerSystemElement`. If you consult the JavaDoc for `ComputerSystemElement,` you see that you can set the additional properties demonstrated in the following code snippet:

Set Additional Properties

```
                ...
                MyHost host1 = MyHost.addInstance(inventoryDataset, new
EntityId(hostsFolder, "host-1"), "host-1", hostsFolder);
                host1.setComputerName(hostname);
        host1.setPrimaryDnsName(dnsName);
        host1.setPrimaryOSType(os);
        host1.setPrimaryRole(role);
        host1.setPrimaryMacAddress(mac);
```

Bulk Configuration Example

Suppose you have a probe that produces an inventory of dozens or even hundreds of elements and associated measurements. For example, suppose the Mock VM Host probe had an inventory of 200 VM's that had 3 CPU's each. Now suppose that you want to turn on the 'CPU Load' metric for each of those 600 CPU's. You do not want to try to do these one at a time through the GUI. Instead you would want to create a bulk configuration template that enabled the 'CPU Load' metric, and then apply that template to your inventory.

Probe Requirements to Support Bulk Configuration

To support bulk configuration with the Template Editor, you must provide the following items in your probe package:

1. A templateDefinition.zip file must be packaged inside of your overall probe zip distribution. This inner zip will contain a fairly complex JSON file that specifies the rules for generating a bulk configuration template for this particular probe.
2. You enable the "Template Editor" button to appear in the Probe Configuration by setting the property *pobc_enable=true* in the <setup> section of your probe.cfx file.

   If you inspect the packaged probe, you see the following structure:

- probe.zip distribution
  - generic (Folder)
  - locales (Folder)
  - pre_install (Folder)
  - templateDefinitions (Folder)
    - templateDefinitions.zip
      - <probe_name>_<version>.json (This is the bulk config template definition)

The structure of the JSON template file is quite complex and would be difficult to generate by hand. The Probe SDK allows you to automate the generation of this file when you build your probe and have it included in the probe distribution.

Process Overview

The SDK includes a Maven plugin called template-definition-generator-plugin that runs when you build your probe. It invokes some java code that you must provide to define the structure of the template you want to provide.

Here is the high level flow of how the Probe SDK builds and packages a bulk config template with your probe.

1. The `mvn install` command is issued to start the Maven build process

2. During the Maven build process, the build plugin com.nimsoft:template-definition-generator-plugin executes. This plugin is provided by the Probe SDK
    a. This plugin generates the <probe_name>_<probe_version>.json bulk configuration template, and also packages it in templateDefinitions.zip
    b. The plugin works by executing some Java code that describes the structure of the probes inventory, and the desired filters. The probe developer must implement this logic. This is what we will explain next.

Here is an overview of what you must implement to enable the build process to produce a valid template:

- Implement a Java class that extends *AbstractTemplateDefinitionCreator*.
- Model your probe's inventory structure using the framework's *GenMonitorType* classes. Build a map that represents parent child relationships and another that represents the type labels.
    - Allow the framework access to these maps by implementing the abstract methods *AbstractTemplateDefinitionCreator.getTypeToLabel()* and *AbstractTemplateDefinitionCreator.getTypeToChildType()*
- Provide detailed information about your probe's inventory and bulk template filter options by overriding the method *TemplateDefinitionCreator.createEntityInfos()*. You construct a list of *CtdEntityInfo* objects that further describe the inventory models relationships and filter options.

A common question probe developers ask at this point is, *"Why do I have to write code to describe my inventory structure? Can't the framework figure it out at runtime?"* Remember that this process executes at probe compile time, not runtime. Since the probe is not yet running and collecting data, there is no way to know what the inventory is going to look like.

Implementing your TemplateDefinitionCreator

Remember, the probe developer must do the following to enable bulk configuration:

1. Create a Java class that extends *AbstractTemplateDefinitionCreator*
2. Define *GenMonitorType* classes that represent the probes inventory and build two maps that represent parent child relationships and the type labels.
3. Allow the framework access to these maps by implementing the abstract methods *AbstractTemplateDefinitionCreator.getTypeToLabel()* and *AbstractTemplateDefinitionCreator.getTypeToChildType()*
4. Provide detailed information about your probe's inventory and bulk template filter options by overriding the method *TemplateDefinitionCreator.createEntityInfos()*. You construct a list of *CtdEntityInfo* objects that further describe the inventory models relationships and filter options.

Extend AbstractTemplateDefinitionCreator

If you generated your probe project from the Maven Archetype as described in Get the Example Source Code, you will find the TemplateDefinitionCreator class defined as follows:

MyMockVMProbeTemplateDefinitionCreator

```
public class MyMockVMProbeTemplateDefinitionCreator extends
AbstractTemplateDefinitionCreator<GenMonitorType>{
...
}
```

Define Basic Inventory Structure with GenMonitorTypes

Since the template generation process occurs at compile time, you do not have access to what the inventory will look like. You must define the inventory based on the knowledge of how the inventory will be constructed at run time. There are two factors which determine the probe inventory structure:

- The types defined in probe_schema.cfg, and
- How those types are related at runtime.

When the inventory was modeled in probe_schema.cfg, the elements were named MyHost, MyVM, MyCPU, and MyMemory

Based on the design work, the inventory model will look like this:

Topology Design

```
RESOURCE
  Hosts(folder)
          MyHost
                  vm(folder)
                          MyVM
                                  CPUs(folder)
                                          MyVMCPU
                                  MyVMMemory
```

You do not need to worry about representing the folders when generating templates, so you must model the following inventory structure:

Topology Simplified

```
RESOURCE
  MyHost
          MyVM
                  MyVMCPU
                  MyVMMemory
```

The first thing you must do in the code is declare GenMonTypes to represent each inventory element the probe creates. Declare the following member variables in the TemplateDefinitionCreator class:

```
    private GenMonitorType myHostType;
    private GenMonitorType myVMType;
    private GenMonitorType myVMCpuType;
    private GenMonitorType myVMMemoryType;
```

Next, initialize the types with the framework provided GenMonitorType.getEnumOrAddAtRuntime(String elementName). To use this method you must pass the name of the inventory element as it is defined in the probe_schema.cfg file. When you modeled the inventory, the elements were named MyHost, MyVM, MyCPU, and MyMemory. You must also define the labels and basic parent child relationships for these

types. You do this with EnumMaps. All of this initialzation can take place in the constructor as shown in the following snippet.

```
    public MyMockVMProbeTemplateDefinitionCreator(String buildDirectory, String
outputDirectory) throws IOException, NimException{
        super();
        // Initialize the graph helper in the parent class
        setGraphHelper();

        // Initialize model information that will be used to generate the template
        // Because the template definition creator class is not used when the probe
        // is actually running, but rather when the probe is built, we must manually
        // populate EnumMaps below

        // Initialize Generic Monitor type Enums that will be used to model
        // parent child relationships of the probes inventory.
        myHostType = GenMonitorType.getEnumOrAddAtRuntime("MyHost");
        myVMType = GenMonitorType.getEnumOrAddAtRuntime("MyVM");
        myVMCpuType = GenMonitorType.getEnumOrAddAtRuntime("MyVMCPU");
        myVMMemoryType = GenMonitorType.getEnumOrAddAtRuntime("MyVMMemory");

        // Initialize the 2 maps we must populate
        myTypeToLabelMap = new EnumMap<GenMonitorType, String>(GenMonitorType.class);
        myTypeToChildMap = new EnumMap<GenMonitorType,
GenMonitorType>(GenMonitorType.class);

        // Model the parent child relationships in our inventory by
        // populating the type to child type EnumMap
        myTypeToChildMap.put(myHostType, myVMType);
        myTypeToChildMap.put(myVMType,myVMCpuType);
        myTypeToChildMap.put(myVMType,myVMMemoryType);

        // Populate our type to label map with the labels we want to see
        myTypeToLabelMap.put(myHostType, "Host");
        myTypeToLabelMap.put(myVMType, "VM");
        myTypeToLabelMap.put(myVMCpuType, "VM CPU");
        myTypeToLabelMap.put(myVMMemoryType, "VM Memory");

    }
```

Implement Map Getters to Allow Framework Access

The base class AbstractTemplateDefinitionCreator defines two abstract methods. You must implement these methods to enable the framework to obtain the parent/child and type/label mappings. Below is the implementation of these methods:

getters

```
    /**
     * Abstract method from AbstractTemplateDefinitionCreator
     * we must implement this as the framework needs access to our
     * type to label mapping to complete the template generation.
     */
    @Override
    protected EnumMap<GenMonitorType, String> getTypeToLabel() {
        return myTypeToLabelMap;
    }
    /**
     * Abstract method from AbstractTemplateDefinitionCreator
```

```
     * we must implement this as the framework needs access to our
     * parent/child mapping to complete the template generation.
     */
    @Override
    protected EnumMap<GenMonitorType, GenMonitorType> getTypeToChildType() {
        return myTypeToChildMap;
    }
```

Implement createEntityInfos()

This is where you provide detailed information about the probe inventory and bulk template filter options. See the following code example:

createEntityInfos()

```
    public List<CtdEntityInfo> createEntityInfos() {

        // Construct CTD entity objects for each of our types
        CtdEntityInfo profile = getProfileEntity();
        CtdEntityInfo resource = getResourceEntity();
        CtdEntityInfo myHostEntityInfo = newCtdEntityInfo(myHostType,
CtdComputerSystem.CTD_ELEMENT_TYPE, CtdTreeIconType.SERVER);
        CtdEntityInfo myVMEntityInfo = newCtdEntityInfo(myVMType,
CtdVirtualSystem.CTD_ELEMENT_TYPE, CtdTreeIconType.VM_ACTIVE);
        CtdEntityInfo myVMCpuEntityInfo = newCtdEntityInfo(myVMCpuType,
CtdProcessor.CTD_ELEMENT_TYPE, CtdTreeIconType.CPU);
        CtdEntityInfo myMemoryEntityInfo = newCtdEntityInfo(myVMMemoryType,
CtdMemory.CTD_ELEMENT_TYPE, CtdTreeIconType.MEMORY);

        // create relationships that model our inventory structure:
        resource.addRelationshipInfo(new
CtdChildRelationshipInfo(myHostType.getName(), CtdRelationshipCardinality.MANY));
        myHostEntityInfo.addRelationshipInfo(new
CtdChildRelationshipInfo(myVMType.getName(), CtdRelationshipCardinality.MANY));
        myVMEntityInfo.addRelationshipInfo(new
CtdChildRelationshipInfo(myVMCpuType.getName()));
        myVMEntityInfo.addRelationshipInfo(new
CtdChildRelationshipInfo(myVMMemoryType.getName()));

        // Specify the filters that will be available on the MyHost object
        myHostEntityInfo.addFilterInfo(FilterBuilder.getLabelFilter(),
                FilterBuilder.getOperationalState(),
                FilterBuilder.getPrimaryDnsFilter(),
                FilterBuilder.getPrimaryIPv4Filter(),
                FilterBuilder.getPrimaryIPv6Filter());

        // Specify the filters that will be available on the MyVM object
        myVMEntityInfo.addFilterInfo(
                FilterBuilder.getComputerNameFilter(),
                FilterBuilder.getLabelFilter(),
                FilterBuilder.getOperationalState(),
                FilterBuilder.getPrimaryDnsFilter(),
                FilterBuilder.getPrimaryIPv4Filter(),
                FilterBuilder.getPrimaryIPv6Filter()
        );

        // Add all the CTD entity object we created to the return data
                 List<CtdEntityInfo> entityInfos = new ArrayList<>();
        entityInfos.add(profile);
        entityInfos.add(resource);
```

```
        entityInfos.add(myHostEntityInfo);
        entityInfos.add(myVMEntityInfo);
        entityInfos.add(myVMCpuEntityInfo);
        entityInfos.add(myMemoryEntityInfo);

        return entityInfos;
    }

    /**
     * Factory method for creating a new instance of CtdEntityInfo
     * based on the passed in GenMonitorType, Element Type, and Icon
     */
    private CtdEntityInfo newCtdEntityInfo(GenMonitorType myType, String
CTD_ELEMENT_TYPE, CtdTreeIconType icon){
        CtdEntityInfo entityInfo = new CtdEntityInfo(myType.getName(),
CTD_ELEMENT_TYPE, myTypeToLabelMap.get(myType));
        entityInfo.setTreeIconOverride(icon);
        entityInfo.setTableDisplayOverride(tableDisplay);
        return entityInfo;
    }
```

Some things to consider in this example are:

- A CtdEntityInfo object was created for each element in the inventory. The framework provides convenience methods getProfileEntity() and getResourceEntity() to obtain the CtdEntityInfo objects for Profile and Resource. However, you must construct you own CtdentityInfo objects for the MyHost, MyVM, MyCPU and MyMemory elements defined in probe_schema.cfg. You see this at lines 6-9. This example makes use of the factory method to instanciate these objects. The code for the factory method is also provided at lines 50-55, showing how to construct a CtdEntityInfo object.
- At lines 12-15 relationships were created to represent how the inventory is structured. This example has the following relationships in the topology:
    - RESOURCE -> MyHost (Many)
    - MyHost -> MyVM (Many)
    - MyVM -> MyVMCPU (Many, but modeled as single. In this example you do not want to provide the ability to filter on CPU.)
    - MyVM -> MyVMMemory (Single)
- At lines 17-32 the filters that will be available on MyHost and MyVM are specified. In this example you do not want to provide any additional filtering on the CPU or Memory elements.
- At lines 34-43 you add all of the CtdEntityInfo objects to a new List and return it.