Practical session 5
(3H)

Java programming

Programing with the VMware Java SDK can be a real pain. Retrieving a simple property on a VMware object requires lot of lines of code. We will try the VI Java API which is a set of Java libraries that sits on top of the existing vSphere SDK Web Services interface.

In order to retrieve properties, perform operations or modify entities within the vSphere API, it is important to understand the vSphere Object Model's structure and relationships. The vSphere API consists of three types of objects:

- **Managed Object**: is a server side object like VMs or Hosts
- **Managed Object Reference**: is a client reference to a Managed Object
- **Data Object**: contains information about a Managed Object and can also be used to transfer data between a client and server

Imagine you want to get the names of all hosts in the inventory, or maybe you want to monitor for changes to the power state of all VMs in the inventory, or you want to keep track of the host that each VM is running on, you have to use the Property Collector.

The PropertyCollector gives you the ability to retrieve properties of the ManagedObject and also monitor the changes of any properties you interested in.

In order to use the PropertyCollector to retrieve informations from Managed Object, you need to specify what data you want to retrieve and this is done with something called PropertyFilterSpec and this object has at least 2 properties:

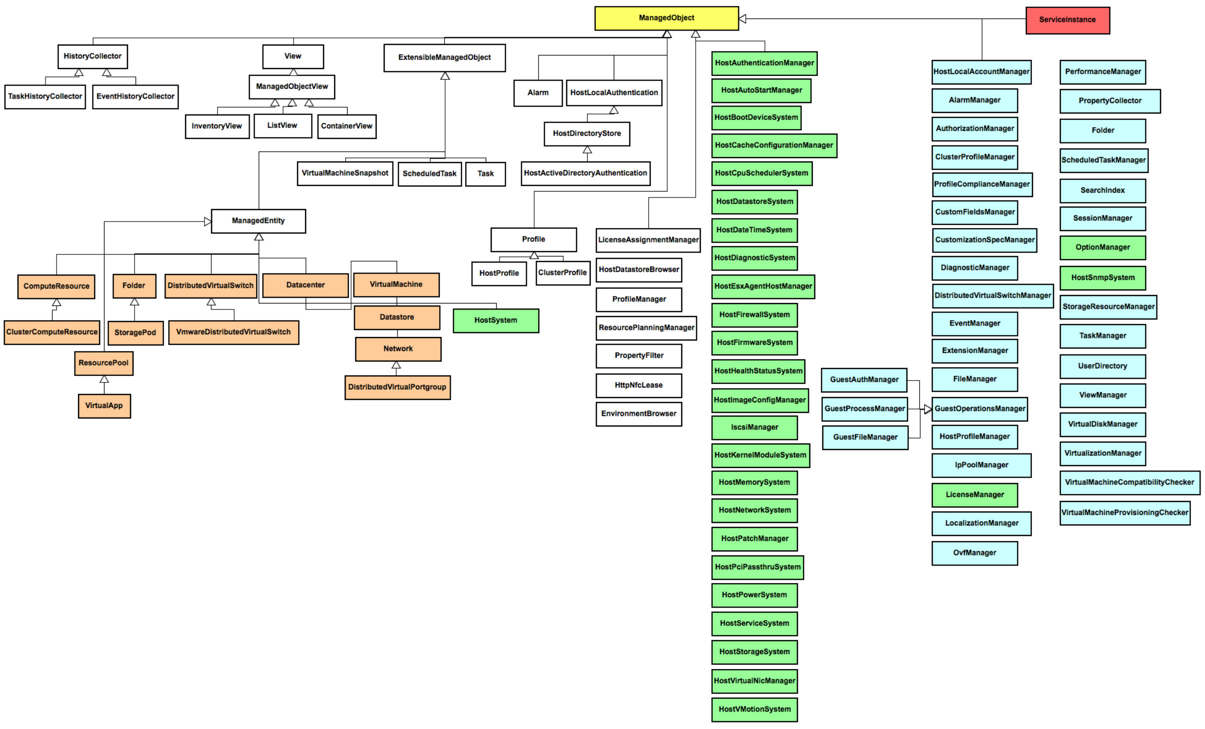
- **PropertySpec** : What property you want to retrieve, the name of a VM for exemple
- **ObjectSpec**: It identifies the starting object in the inventory
- **TraversalSpec**: (optional): Define how to traverse the inventory

And those objects have other objects as parameter,and you need to create. Obviously, you see that it requires many lines of code before retrieving the name of a VM or whatever properties you want to get from your vCenter.

vijava (Virtual Infrastructure API) is an open source project created by Steve Jin from VMware R&D. It aims to simplify the use of VI SDK and improve the performance.

This API encapsulate the PropertyCollector behind Java methods and provide a whole bunch of ManagedObject that will allow you to manipulate vmware objects without any effort.

The picture below represent the object architecture that will be very helpful to code you program and you will see further in this article that I manipulate those objects to retrieve specific properties.



There is no really documentation, but you can read : http://grepcode.com/snapshot/repo1.maven.org/maven2/com.cloudbees.thirdparty/vijava/5.5-beta/

You fin many samples here : https://github.com/Juniper/vijava/tree/master/src/com/vmware/vim25/mo/samples

You can test YAVIJA, the new version of vijava : http://www.yavijava.com https://github.com/yavijava/yavijava

```
You can compile like this :
javac ListHosts.java -cp “./dom4j1.6.1.jar:./vijava55b20130927.jar”

And run :
java  -cp "./dom4j-1.6.1.jar:./vijava55b20130927.jar:." ListHosts

With YAVIJAVA :
javac  -cp "./dom4j-1.6.1.jar:./yavijava-6.0.05.jar:./log4j-1.2.17.jar:."
ListHosts.java
Run
java  -cp "./dom4j-1.6.1.jar:./yavijava-6.0.05.jar:./log4j-1.2.17.jar:."
ListHosts
```

1) Connect to the ServiceInstance of your ESXi

```
public ServiceInstance Initialisation(
            String url,
            String username,
            String password)
            throws RemoteException, MalformedURLException {
      ServiceInstance si =
            new ServiceInstance(new URL(url), username, password, true);
      return si;
}
```

This is the method you can use to connect your ESXI servive. The initialization method takes 3 parameters which are:

- **Url**: this is the URL of the ESXi service and it look like this
    - `https://192.168.1.1/sdk/vimService`
- **Username**: nothing special
- **Password**: nothing special

This method will return a ServiceInstance object which will be used to have access to your VMware object tree (see the picture).

2) **Retrieve ManagedEntity from Inventory Navigator**

```
public ManagedEntity[] RetreiveVM(ServiceInstance si)
      throws InvalidProperty, RuntimeFault, RemoteException{
      ManagedEntity[] mes =
      new InventoryNavigator(
            si.getRootFolder()).searchManagedEntities("VirtualMachine");
      return mes;
      }
```

This method takes the ServiceInstance object that we had in the previous section. Here we have to create a rootFolder object from our ServiceInstance object.

And next we can already retrieve ManagedObject (VirtualMachine in my exemple) with the searchManagedEntities method and fill the mes[] table with all the VMs of your infrastructure.

From the InventoryNavigator object, you can use:

- searchManagedEntities(« VirtualMachine »)
- searchManagedEntities(« HostSystem »)
- searchManagedEntities(« Datacenter »)
- and some other that i am not aware of…

With these methods, you can already retrieve VMs from your vCenter. The next will be to retrieve information of the VM and you will see that in the next section.

3) **Convert ManagedEntity object and retrieve VM properties**

```
public void setSelectedVM(int index) {
      VirtualMachine vm = (VirtualMachine) mes[index];
      System.out.println(
      "Guest OS:"+vm.getSummary().getConfig().guestFullName);
}
```

First of all, you will have to convert the « mes » object into a « virtualMachine » object. Then you are free to exploit the hundreds of properties of the VM object.

4) Question 1

Try the example ListHost

5) Question 2

List all VMs, print VM configuration

6) Question 3

Retrieve VM hardware properties :

VirtualDisk
VirtualEthernetCard
MAC address

7) Question 4

Implement, the VM powerOn, powerOff, suspend, resume.

8) Question 5

We'll be working with snapshots, creating snapshots.
You need to perform the following steps:

- Get a new ServiceInstance
- Search for the virtual machine
- Create a snapshot task

Use the createSnapshot_Task method in VirtualMachine

The call to createSnapshot_Task immediately returns with a Task object. Using this Task instance, you can monitor what's the progress of the task. It is done:

- Getting a TaskInfo instance from the task object
- Checking for the current state

You can do it in a loop like this:

```
TaskInfoState state;
do {     Thread.sleep(1000);     state = task.getTaskInfo().getState();
   System.out.println("State="+state); }
while (state != TaskInfoState.error && state != TaskInfoState.success);
```

8) Question 6

Your ESX system collects data at runtime about performance for all aspects of the system (CPU, disk, memory, network, and so on). The data in the counters is accessible through the PerformanceManager class.
You can get an instance of the PerformanceMangager like this:

```
ServiceInstance si = new ServiceInstance(
      new URL(url), USER_NAME, PASSWORD, true);
      PerformanceManager perfMgr = si.getPerformanceManager();
```

The performance manager has a getPerfCounters method which will return a list of all available counters:

```
PerfCounterInfo[] perfCounters = perfMgr.getPerfCounter();
for (int i = 0; i < perfCounters.length; i++) {
PerfCounterInfo perfCounterInfo = perfCounters[i];
String perfCounterString = perfCounterInfo.getNameInfo().getLabel()
      + " (" + perfCounterInfo.getGroupInfo().getKey() + ")
      in
      " + perfCounterInfo.getUnitInfo().getLabel()
      + " (" + perfCounterInfo.getStatsType().toString() + ")";
      System.out.println(perfCounterInfo.getKey() + " : " +
perfCounterString); }
```

This will return a list like this:

```
1 : Usage (cpu) in Percent (rate)
...
336 : Queue write latency (disk) in Millisecond (absolute)
337 : Physical device command latency (disk) in Millisecond (absolute)
338 : Kernel command latency (disk) in Millisecond (absolute)
```

perfCounterInfo.getKey() returns the key with which you can identify the counter to get some information about the performance of the system.

A query for performance counter is done using the PerformanceManager. In order to query for counters you need to create an instance of PerfQuerySpec which defines what we'll be querying. You need to define the following:

- which entity should be queried (in our case the host)
- which metrics should be queried (e.g. the counter number XX (active memory in KB)
- the query interval: The interval (sampling period) in seconds for which performance statistics are queried. There is a set of intervals for historical statistics. To retrieve the greatest available level of detail, the provider's refreshRate may be used for this property

Then let's get the interval ID:

```
PerfProviderSummary summary = perfMgr.queryPerfProviderSummary(host);
int perfInterval = summary.getRefreshRate();
System.out.println("Refresh rate = " + perfInterval);
```

Now we can create a PerfQuerySpec:

```
PerfMetricId pmis ….
PerfQuerySpec qSpec = new PerfQuerySpec();
qSpec.setEntity(host.getMOR());
qSpec.setMetricId(pmis);
qSpec.intervalId = perfInterval;
```

and retrieve value :

```
PerfEntityMetricBase[] pembs =
      perfMgr.queryPerf(new PerfQuerySpec[] { qSpec });
```

Some things like `pembs[i].getValue();`