ARRAY PRODUCT

Quick Start Guide

NOVEMBER 17, 2020

MICHAEL WALKER

Version 1.0.2

QUICK START GUIDE FOR ARRAY PRODUCT INFORMATION

TABLE OF CONTENTS

Introduction

WHAT IS A QUICK START GUIDE?

SMI-S is 2516 pages of reading spread across 8 books, plus it references another 14 or so DMTF profiles which amount to another 660 pages of reading. So, the question is where do you start? We have come up with a series of Quick Start Guides that are designed to help you get started by illustrating how to find useful SMI-S information in mock servers (mock ups of SMI-S server implementations). The Quick Start Guides don't illustrate EVERYTHING in the 3176 pages, but they give you a head start at finding some important items in SMI-S.

We currently have quick start guides for:

1. The Interop Namespace - What is it and what does it tell us?
2. Performance Information - Where do I find performance information in an SMI-S Server?
3. Capacity Information - Where do I find storage capacity information in an SMI-S Server?
4. Hardware Information - Where do I find hardware information in an SMI-S Server?
5. Product Information - Where do I find product information in an SMI-S Server?
6. Software Information - Where do I find software information in an SMI-S Server?

TOOL USED FOR THE QUICK START GUIDE ILLUSTRATION

The tool used for illustrating how to find information in SMI-S is the pywbemcli (part of pywbemtools). It is a command line interface for accessing any WBEM Server. It uses pywbem, an interface for python program access to any WBEM Server. The pywbemtools (and the pywbemcli) and pywbem are python programs that use a set of python packages. Pywbem and the pywbemtools are actively being maintained and are available on Github.

We will be using the latest version of the pywbemcli in these guides. You can find documentation on the pywbemcli at the following website:

https://pywbemtools.readthedocs.io/en/latest/

THE MOCK IMPLEMENTATIONS

The mock implementations mock selected autonomous profiles and some of their component profiles in SMI-S 1.8.0.

We currently have mock ups for the following autonomous profiles:

1. The SNIA Server Profile
2. The DMTF WBEM Server Profile
3. The Array Profile
4. The NAS Head Profile

And we plan on doing a Fabric (and Switch) mock up.

We chose to do mock ups of the SMI-S 1.8.0 versions of these profiles to illustrate differences between 1.8.0 and 1.6.1. We don't mock everything that is new in 1.8.0, but we do highlight some key changes ... like the DMTF WBEM Server profile, new indications and Advance Metrics (performance) for Arrays.

HOW A GUIDE WORKS

The guide is a sequence of text explaining what we are looking for, followed by the command to obtain the information, followed by command output and then text that explains the output.

THE QUICK START GUIDE FOR ARRAY PRODUCT

In this guide we will be exploring product information provided by an SMI-S Server for an Array. The mock for the Array supports SMI-S 1.8.0.

This 3-page document highlights information that can be found in about 10 pages of SMI-S, the Physical Package profile in the common profiles book of SMI-S.

In this script we will be working with pywbem 1.1.1 and version 0.8.0 of pywbemtools (pywbemcli).

So, let's begin. First, we go to our virtual environment for mocks:

```
C:\Users\FarmerMike> workon mocks
(mocks) c:\Users\FarmerMike\devenv>
```

We are now in our virtual environment for mocks.

We will be working with a mock server that supports an SMI-S 1.8.0 Array. So, we need to establish a connection to the Array mock up:

```
(mocks) c:\Users\FarmerMike\devenv> pywbemcli -o table -n ArrayMock
Enter 'help' for help, <CTRL-D> or ':q' to exit pywbemcli or <CTRL-r> to search history,
pywbemcli>
```

In our command, I requested the default format of output to be in “table” format (-o table) and name the mock that I want (-n ArrayMock). The command worked and we get a pywbemcli prompt to start entering commands on the ArrayMock.

PRODUCTS

The elements that reports product information are instances of CIM_Product.

So, we start by looking for classes that report product information (CIM_Product)

```
pywbemcli> instance enumerate CIM_Product --pl IdentifyingNumber,name,Vendor,Version
Instances: CIM_Product
+-----------------------------------+-----------+---------------------------+-------------+
| IdentifyingNumber                 | Name      | Vendor                    | Version     |
|-----------------------------------+-----------+---------------------------+-------------|
| "ACME+CF2A5091300089" | "ACME" | "ACME Corporation" | "1.1.0"     |
+-----------------------------------+-----------+---------------------------+-------------+
pywbemcli>
```

The array mock is only showing one product: The ACME product, version 1.1.0.

WHAT THE PRODUCT REPRESENTS

The next question to answer is what part of the Array does the product represent.

We will determine this by the following command to determine logical elements supported by the product.

```
pywbemcli> -o mof instance shrub CIM_Product.?
CIM_Product.IdentifyingNumber="ACME+CF2A5091300089",Name="ACME",Vendor="ACME
Corporation",Version="1.1.0"
 +-- GroupComponent(Role)
    +-- CIM_ProductPhysicalComponent(AssocClass)
       +-- PartComponent(ResultRole)
          +-- CIM_PhysicalPackage(ResultClass)(1 insts)
             +-- /:CIM_PhysicalPackage.CreationClassName="CIM_Chassis",Tag="ACME+CF2A5091300089"
pywbemcli>
```

There are a couple of things to notice with this command output. If there is only one product to select from you are not prompted for a selection. The command automatically generates the results for the one item.

The shrub consists of a CIM_ProductPhysicalComponent to a physical package (a CIM_Chassis).

We can determine what the chassis supports by issuing the following command:

```
pywbemcli> -o mof instance shrub CIM_PhysicalPackage.?
Pick Instance name to process
0: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_Chassis",Tag="ACME+CF2A5091300089"
1: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+"
2: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+ACME+0_0_0"
3: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+ACME+0_0_1"
4: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+ACME+0_0_2"
5: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+ACME+0_0_3"
6: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+ACME+0_0_4"
```

```
7: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+ACME+0_0_5"
8: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+ACME+0_0_6"
9: root/cimv2:CIM_PhysicalPackage.CreationClassName="CIM_PhysicalPackage",Tag="ACME+CF2A5091300089+0_0+ACME+0_0_7"
Input integer between 0 and 9 or Ctrl-C to exit selection:
```

Our physical package is the first physical package, so we select 0:

```
Input integer between 0 and 9 or Ctrl-C to exit selection: 0
CIM_PhysicalPackage.CreationClassName="CIM_Chassis",Tag="ACME+CF2A5091300089"
 +-- GroupComponent(Role)
 |   +-- CIM_Container(AssocClass)
 |       +-- PartComponent(ResultRole)
 |           +-- CIM_PhysicalPackage(ResultClass)(1 insts)
 |               +-- /:CIM_PhysicalPackage.~,Tag="ACME+CF2A5091300089+0_0+"
 +-- PartComponent(Role)
 |   +-- CIM_ProductPhysicalComponent(AssocClass)
 |       +-- GroupComponent(ResultRole)
 |           +-- CIM_Product(ResultClass)(1 insts)
 |               +-- /:CIM_Product.IdentifyingNumber="ACME+CF2A5091300089",Name="ACME",Vendor="ACME
Corporation",Version="1.1.0"
 +-- Antecedent(Role)
     +-- CIM_SystemPackaging(AssocClass)
         +-- Dependent(ResultRole)
             +-- CIM_ComputerSystem(ResultClass)(1 insts)
                 +-- /:CIM_ComputerSystem.~,Name="ACME+CF2A5091300089"
pywbemcli>
```

The last association CIM_SystemPackaging indicates that the physical package is associated with a ComputerSystem.

We can look at the ComputerSystem to see what it is using the following command:

```
pywbemcli> instance get CIM_ComputerSystem.? --pl CreationClassName,Name,Dedicated
Pick Instance name to process
0: root/cimv2:CIM_ComputerSystem.CreationClassName="CIM_ComputerSystem",Name="ACME+CF2A5091300089"
1: root/cimv2:CIM_ComputerSystem.CreationClassName="CIM_ComputerSystem",Name="ACME+CF2A5091300089+SP_A"
2: root/cimv2:CIM_ComputerSystem.CreationClassName="CIM_ComputerSystem",Name="ACME+CF2A5091300089+SP_B"
Input integer between 0 and 2 or Ctrl-C to exit selection:
```

Our system is the first system, so we select 0:

```
Input integer between 0 and 9 or Ctrl-C to exit selection: 0
Instances: CIM_ComputerSystem
+--------------------------------+-----------------------------------+---------------------------------------+
| CreationClassName              | Name                              | Dedicated                             |
|--------------------------------+-----------------------------------+---------------------------------------|
| "CIM_ComputerSystem" | "ACME+CF2A5091300089" | 3 (Storage), 15 (Block Server) |
+--------------------------------+-----------------------------------+---------------------------------------+
```

```
pywbemcli>
```

And we see the ComputerSystem has a Dedicated property value of 3 and 15. This means the Chassis is the Chassis for the Array.