

## 5 CCS Scripts to Jump Start Your NetMRI CCS Training

### Intro

[NetMRI](#) is the world's leading multi-vendor network automation solution. It provides Infoblox Switch Port Manager and Automation Change Manager which gives you the ability to perform change and configuration management as well as compliance enforcement.

### What is CCS

CCS is a proprietary scripting language for executing sequences of command-line interface (CLI) commands on NetMRI-supported network devices. This allows you to perform automation of configuration tasks. If you know Cisco IOS, writing CCS scripts is very straightforward. This post presumes some prior knowledge of Cisco IOS or any other CLI interface for devices such as Extreme, Juniper or other vendors supported by NetMRI. Some previous structured programming knowledge will also be helpful.

While CCS is not as powerful or flexible as Perl, the learning curve is not as steep. CCS allows you to quickly develop useful jobs that can run across hundreds of devices in the managed network.

### Why Use Scripting?

- Scripts:
  - Automate changes to infrastructure devices
  - Ensure consistent changes
  - Can be scheduled to run at specific times
  - Can be triggered to respond to specific events
  
- Scripting reduces the risk of errors being made during execution
  - No fat fingering
  - No instant mistakes

Now let's jump into it

# Script 1 – “The Basics”

## NetMRI Version – any

To get you familiar with the CCS scripting language in NetMRI, just “The Basics” to get you started.

### The anatomy of a CCS script

```
Script-Filter:
  true
#####
Action:
  Config SNMP
Action-Commands:
  config t
  snmp-server READ infoblox RO
  snmp-server WRITE netmri RW
  end
  wr mem
```

The diagram illustrates the structure of a CCS script with three callout boxes:

- Script-Filter:** A green box explains that 'true' means 'no matter what device it is. Log into it'.
- Action:** A green box explains that 'Config SNMP' is the 'Action Name' and 'means give me a unique name'.
- Action-Commands:** A green box explains that this section is where you execute commands as if you were at an 'enable' prompt in Cisco or any other vendors.

Download Link – [PS Training SNMP v1.ccs](#)

# Script 2 – “Multi-Vendor”

## NetMRI Version – any

To show the power of NetMRI multi-vendor support for updating SNMP Community strings.

```
Script-Filter:
  $vendor in ["Cisco","HP"]

#####
Action:
  Config SNMP
Action-Commands: {$Vendor eq "HP"}
  config t
  snmp-server READ infoblox RO
  snmp-server WRITE netmri RW
  end
  wr mem
Action-Commands: {$Vendor eq "Cisco" and $sysdescr like /IOS/}
  config t
  snmp-server community infoblox RO
  snmp-server community netmri RW
  end
  wr mem
#####
```

This now means, only run the script on Devices where Vendor equal Cisco or HP. \$vendor is one of our "Well Known" variables

Configlets for multi-vendor and OS types

Now we have one Action, with multiple Action Commands. But we added something a little special. Action-Commands with " {" }" This evaluates what's in the curly brackets and if it's "True" it runs the Action-Commands that follow.

Now we did with both \$Vendor and \$sysdescr. Here is the "magic" it will only run on Cisco Devices that have IOS, but SKIP NX-OS. So if we wanted to we can add another action Action-Commands for NX-OS

Download Link - PS Training SNMP v1 5.ccs

The beauty of this script is that it's simple but so powerful imagine having one script that can change all the Community Strings on all your L2/L3 devices.

# Script 3

## Find and Replace

Now say you want to replace all your SNMP Community strings to a new standard, here is an example.

## Find and Remove

```

Script-Filter:
$vendor eq "Cisco" and $sysdescr like /IOS/
#####
Action:
Show SNMP
Action-Commands:
SET: $updatemade = "no"
sh run | inc snmp-server
Output-Triggers:
Parse Output
#####
Trigger:
Parse Output
Trigger-Variables:
$snmpcom string
Trigger-Template:
snmp-server community [[$snmpcom]]
Trigger-Commands: { $updatemade eq "no"
config t
SET: $updatemade = "yes"
Trigger-Commands:
no snmp-server community $snmpcom
#####
Action:
Add New Community
Action-Commands: { $updatemade eq "yes"
snmp-server community infoblox RW
end
copy run start lr
  
```

Now we moved \$vendor and \$sysdescr to Script-Filter, so what does this mean? Before NetMRI open a Session to a device it will check it's Vendor and SysDescr to see if it's a "Cisco" device and if the OS is "IOS", then we log into it.

This is the output from the show command:  
snmp-server community public RO  
snmp-server community infoblox RW  
snmp-server community sifbaksh RO  
We now pass that entire output to a Trigger called Parse Output

Now we will run a show command and parse the output in a NetMRI CCS script.

Parse Output is just the name of the Trigger that get's called from Output-Triggers

Create a variable to capture something in

Under Trigger-Template this is where we want to capture the SNMP Community String and then, use the community string to remove it

Under Trigger-Template this is where we want to capture the SNMP Community String and then, use the community string to remove it

Now that we have captured the community string, we remove it by using the command "no logging \$snmpcom"

Once all the snmp-server community strings have been removed We add the one we want

Trigger is basically a loop It will loop through all the Show command output

Now you can see the possibility with NetMRI Trigger Commands and yes you can pass one set of Trigger Commands to another.

The table below shows some Trigger Variables examples from the [CSS Supplement Guide](#).

extractPattern	Corresponding Regular Expression
String	/[^\r\n]+/
Word or Id	/\w+ /
Int or Integer	/\d+ /
Double or Float	/\d+\.\d+ /
Datetime	/\d{2,4}-\d{1,2}-\d{1,2} \d{1,2}:\d{1,2}:\d{1,2} /
Phoneno	/\d{3}[-]?[-]? \d{3}-\d{4} /
Zipcode	/\d{5}(-\d{4}) /
Email	/[^\w\.-]+@[^\w\.-]+ /
Ipaddress	/\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3} /
Regular expression	/Any valid Perl 5 regular expression /

# Find, Remove and Keep

Almost the same as above, you can keep some of the configs that you want.

## Find/Remove and KEEP

```
Script-Filter:
$vendor eq "Cisco" and $sysdescr like //IOS/

#####
Action:
  Show SNMP

Action-Commands:
  SET: $updatedmade = "no"
  sh run | inc snmp-server

Output-Triggers:
  Parse Output

#####
Trigger:
  Parse Output

Trigger-Variables:
  $snmpcom string

Trigger-Template:
  snmp-server community [[${snmpcom}]

Trigger-Commands: { $updatedmade eq "no" and $snmpcom not in ["infoblox","sif"]}
  config t
  SET: $updatedmade = "yes"

Trigger-Commands: { ${snmpcom} not in ["infoblox","sif"]}
  no snmp-server community $snmpcom

#####
Action:
  Add New Community

Action-Commands: { $updatedmade eq "yes"
  snmp-server community infoblox RW
  snmp-server community sif RW
  end
  copy run start lr
```

Now we will run a show command and parse the output in a NetMRI CCS script.

Now we only remove the snmp community string that is "not in" the list infoblox,sif.

Now we only remove the snmp community string that is "not in" the list infoblox,sif. So anything else will be removed

Trigger is basically a loop It will loop through all the Show command output

Now let's take the above script and change a few things, like the show command and the Trigger Template, Trigger Variable and the Trigger Command.

# A few changes and a New Script

## Tweak the Framework for something else

```
Script-Filter:
  $vendor eg "Cisco" and $sysdescr like /IOS/
#####
Action:
  Show SNMP

Action-Commands:
  SET: $updatedmade = "no"
  sh run | inc logging

Output-Triggers:
  Parse Output

#####
Trigger:
  Parse Output

Trigger-Variables:
  $snmpcom ipaddress

Trigger-Template:
  logging [[$snmpcom]]

Trigger-Commands: { $updatedmade eq "no" and $snmpcom not in ["10.10.10.1"]
  config t
  SET: $updatedmade = "yes"

Trigger-Commands: {$snmpcom not in ["10.10.10.1"]}
  no logging $snmpcom

#####
Action:
  Add New Community

Action-Commands: { $updatedmade eq "yes"}
  logging 10.10.10.1
  end
  copy run start \r
```

Now we will run a show command and parse the output in a NetMRI CCS script.

Now we changed the show command to "sh run | inc logging"  
Change the variable type to "ipaddress" from "string" cause we want to capture an IP address.  
We change the Template cause what we want to match has changed

We also change the not in to "10.10.10.1" the IP address that we want to keep and then remove the logging.

Trigger is basically a loop  
It will loop through all the Show command output

Yes that's how easy it is to change the script for something else. Now go think about all the possibilities of the other show commands you can run and do something with it.

# IP Helper Script

Now we all have IP Helpers or something on each Interface that we would like to find and change. If you are looking at rolling out ISE and something similar to it you will find this script very useful.

## IP helper-address Script Explained!

Script-Filter:  
\$Vendor eq "Cisco" and \$Type in ["Router","Switch-Router","VoIP Gateway","Switch"] and \$SysDescr like /IOS/

#####  
Action:  
Find Interfaces

Action-Description:  
Find valid interfaces in order to process them for IP Helper addresses.

Action-Commands:  
SET: \$UpdateMade = "no"  
SET: \$FoundHelper = "no"  
show ip int brief

Output-Triggers:  
Process Interfaces

#####  
Trigger:  
Process Interfaces

Trigger-Description:  
Find valid interfaces to check for helpers - An interface that has an ip address and is "up"

Trigger-Variables:  
\$IntName Aw+Id+(Vd{1,2})\Vd{1,2}\Vd{1,2}\Vd{1,2}\Vd{1,2}??  
\$IntIP Ad{1,3}\Vd{1,3}\Vd{1,3}\Vd{1,3}

Trigger-Template:  
[[\${IntName}]]s+[[\${IntIP}]]s+w+s+w+s+up+s+up

Trigger-Commands:{\$UpdateMade eq "no" and \$IntName not in ["channel1","ap0","NV10"]  
sho run interface \$IntName  
Trigger-Commands:{\$UpdateMade eq "yes" and \$IntName not in ["channel1","ap0","NV10"]  
do sho run interface \$IntName

Output-Triggers:  
Check helper 1  
Check helper 2

Trigger:  
Check helper 1

Trigger-Template:  
ip helper-address 10.134.2.100

Trigger-Commands:{\$UpdateMade eq "no")  
config t  
SET: \$UpdateMade = "yes"

Trigger-Commands:  
interface \$IntName  
no ip helper-address 10.134.2.100  
ip helper-address 10.150.98.100  
ip helper-address 10.132.101.100

#####  
Trigger:  
Check helper 2

Trigger-Template:  
ip helper-address 10.132.2.100

Trigger-Commands:{\$UpdateMade eq "no")  
config t  
SET: \$UpdateMade = "yes"

Trigger-Commands:  
interface \$IntName  
no ip helper-address 10.132.2.100  
ip helper-address 10.132.101.100  
ip helper-address 10.150.98.100

#####  
Action:  
End and Write Mem

Action-Description:  
End config mode and write memory only if a change was made

Action-Commands:{\$UpdateMade eq "yes")  
end  
write mem

Script filter  
We only execute this script on Routers, Switch-Router, Switch and VoIP Gateway if they are running IOS only!  
Our First command we run on the devices "show ip int brief", this gives us an output of all the interface configured on the connected device.  
We take that output and parse it for to get just each interface to run a show command to find out if "ip helper-address" is configure on that interface or not.

The first variable \$IntName this is how we capture the device name (i.e. interface Fa01 - Fa1/1 - Gi2/0/1..etc)  
Trigger-Template is matching all the interface that are up/on on the device via the "show ip int brief".  
Once we grab the interface we will run a "show run interface \$IntName" (\$IntName if the "interface" we captured from the output) command to parse that output of each "interface" looking for the First "ip helper-address" using the Check helper 1 and then the same of the 2<sup>nd</sup> using Check helper 2.  
Reason you have 2 show commands, one with "do" is because we are in configure t when we loop though the next interface.  
We had some issues with "channel,ap0 and NV10" interfaces, so this syntax will skip over this interfaces, you can append more interfaces in the future.

We will loop through every Interface from the "show ip int brief" Then go to Trigger Check helper 1 then Check helper 2 Then loop back to This Trigger Then start with the show run int \$IntName. Over again.

Check helper 1, is using the Trigger-Template of ip helper-address 10.134.2.100 meaning if I see that statement under an interface(that's consider a match) then run the Trigger-Commands. In this case remove "no ip helper-address 10.134.2.100" then add the 2 we want just in case the other one is not there.  
Same thing with Check helper 2.

Trigger is basically a loop It will loop through all the Show command output

Once all the "ip helper-address" have been changed we save it.