

USER MANUAL



IxNetwork - Low Level API

Contents

[1. **Overview** 2](#_Toc512647899)

[2. **API Tree Structure** 2](#_Toc512647900)

[3. **IxNetwork Hierarchical Structure** 2](#_Toc512647901)

[4. **Installation of IxNetwork Low Level TCL** 3](#_Toc512647902)

[4.1 Windows-based 3](#_Toc512647903)

[4.2 LINUX-based 4](#_Toc512647904)

[5. **Configuring BGP by using LLAPI in TCL and Python** 5](#_Toc512647905)

[5.1 Import ixNetwork Tcl API Package 5](#_Toc512647906)

[5.2 Connect to the Chassis 5](#_Toc512647907)

[5.3 Protocol Configuration 6](#_Toc512647908)

[5.4 Start Protocol and Check Statistics 12](#_Toc512647909)

[5.5 L2/L3 Traffic Configuration/Apply/Start Section 15](#_Toc512647910)

[5.6 Packet Captures 16](#_Toc512647911)

[5.7 Apply Start Traffic 17](#_Toc512647912)

[5.8 Retrieving Statistics 18](#_Toc512647913)

[5.9 Generate Scriptgen 20](#GenerateScriptgen)

[6. **References** 21](#_Toc512647915)

[7. **Support** 21](#_Toc512647916)

# [1. Overview:](#ref_TOC)

Low level API can be used to configure both classic and NGPF GUI. Low level API can be executed in TCL,Perl and Python. This document will give you the idea about the installation of IxNetwork on windows and linux machines and also about the creation of protocol-stack configuration model by executing commands in TCL and Python script using low level API.

# 2. [API Tree Structure](#ref_TOC2):

The internal architecture and implementation of the IxNetwork automation stack is as shown in the following figure. This automation stack shows all the API offerings of IxNetwork.

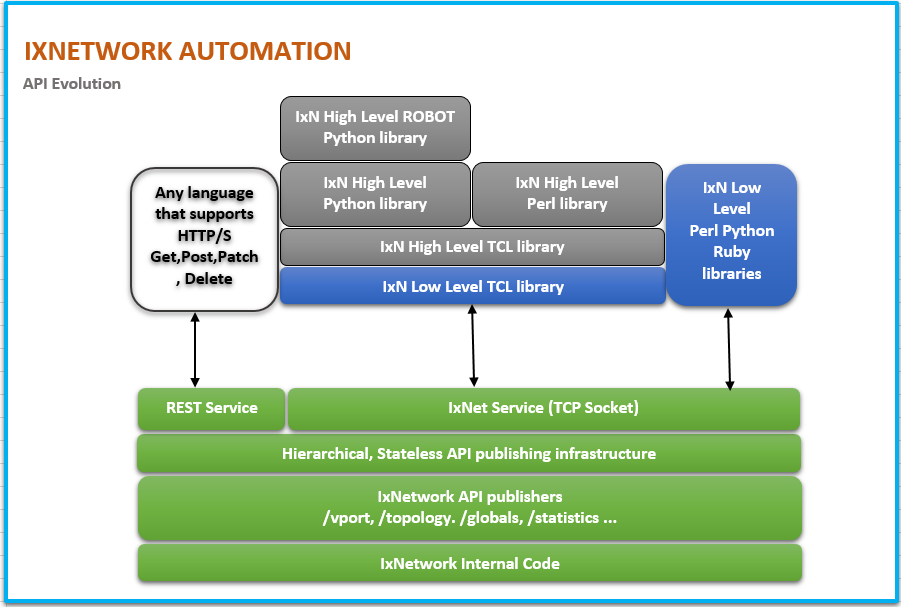


Figure 2.1: IxNetwork API Stack

# 3. [IxNetwork Hierarchical Structure:](#ref_TOC3)

Any configuration that you build in the IxNetwork GUI appears as a hierarchical-tree like data-model to the IxNetwork APIs. Nodes of the hierarchical tree are also known as objects, and properties of the nodes are known as attributes of the node or object.

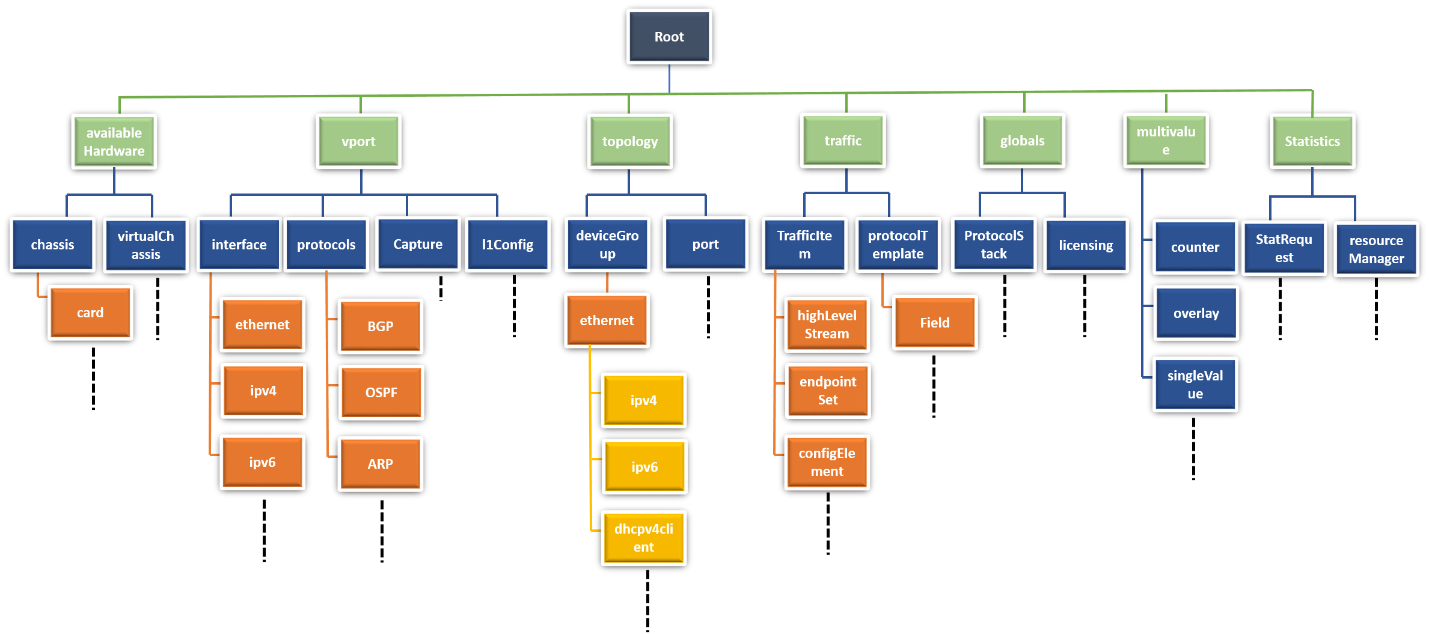


Figure 3.1: Hierarchical-tree Data Model of LLAPI

All IxNetwork TCL commands start with the keyword “ixNet.” The generic format of the IxNetwork TCL commands is “**ixNet subcommand args\*.**”

Description of the basic subcommands used commonly in the LLAPI are:

|  |  |
| --- | --- |
| SubCommands | Description |
| ixNet getRoot | This command returns the id "root" node of the IxNetwork API tree. The return value of the API is a string such as "::ixNet::OBJ-/". |
| ixNet help <objref> <propname> | This command displays basic help for the listed object reference <objref>, its property <propname>, or command. |
| ixNet getList <objref> <listName> | This command takes the node id and child type as an argument, and returns a list of children of that type. |
| ixNet getAttribute <objref> -<attributeName> | This command retrieves the value of an attribute -<attributeName> from the given object reference <objref>. |
| ixNet setAttribute <objref> -<attributeName> <value> | This command sets the value of a specified attribute (<attributeName>) of an object <objref>. |
| ixNet setMultiAttribute <objref> -<attributeName1> <value1> -<attributeName2> <value2> … | This command sets the value of several attributes (<attributeName1>, <attributeName2>, ...) of an object (<objref>) at one shot. |
| ixNet commit | This command applies all changes into the configuration that you have made by using the ixNet setAttribute, ixNet add, or ixNet remove command |
| ixNet add <objref> <childType> | This command adds an object to an object's list property. |
| ixNet remapIds <tcl list of objrefs and temporary objrefs> | This command takes the list of temporary object-id and input and returns list of permanent object id as output, provided have run the ixNet commit to complete the change(s). |
| ixNet execute <function> <arg\*> | This command runs a function <function> with argument(s) <arg\*>. |

Table 3.1: Description of the basic commands used in LLAPI

# 4. [Installation of IxNetwork Low Level TCL](#ref_TOC4):

Environment variables are (case-sensitive) named values that are provisioned and maintained by the operating system, which is used by dynamic programming languages like TCL, Python, Perl, and Ruby to get informed about the operating environment, OS type, and location of library paths.

## 4.1 [Windows-based](#ref_TOC4a):

The environment variable i.e. TCLLIBPATH should be populated with the IxNetwork's TCL Library path. Set the variables in environment variable prompt as follows:

**Variable name**: TCLLIBPATH

**Variable value**: {C:/Program Files (x86)/Ixia/IxNetwork/<IxNetwork install path>/API/TCL/IxTclNetwork}



Figure 4.1.1: Setting environment Variable TCLLIBPATH

Type “**package req IxTclNetwork**” on the Command Prompt window. It should return the IxTclNetwork version installed on your computer.

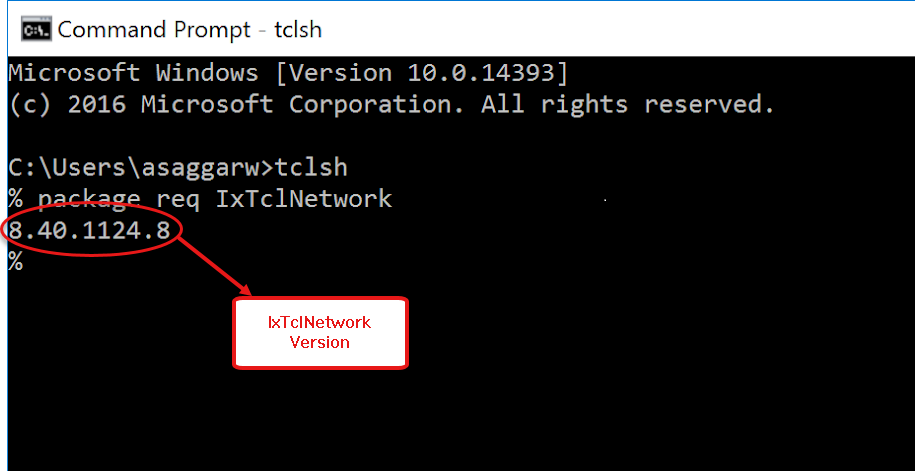


Figure 4.1.2: TCL Interpreter Returned Proper IxNetworkTcl Version

## 4.2 [LINUX-based](#ref_TOC4b):

Setting environment can be done from a shell script. Following are bare necessary steps for setting up an environment to run the low-level TCL script:

1. Open a file by using an editor, such as, vi env.sh.

2. Write the following commands in the file:

export TCLLIBPATH=“<IxNetwork Library Path>/ixnetwork/8.40.1077.17/lib/IxTclNetwork/”

3. Close the file.

4. Source the file: source env.sh.

Open a TCL shell. Invoke the low-level TCL packages successfully as shown:

|  |
| --- |
| [localhost@ixnetwork ~]$ tclsh  % package req IxTclNetwork  8.40.1077.17  % exit [localhost@ixnetwork ~]$ |

# **5.** [Configuring BGP by using LLAPI in TCL and Python:](#ref_TOC5)

The follow section is intended to give you the basics of configuring BGP in a TCL and Python script using the low-level API. The steps are laid out in the order of execution. Context is also provided through screen shots of the IxNetwork GUI that show the outcome of the commands

Similarly other protocols can also be implemented using the same logic.

## 5.1 [Import ixNetwork Tcl API Package](#ref_TOC5a):

In order to use the IxNetwork Tcl and Python API, the Tcl and Python interpreter needs to be told where to find the definitions of the API. These commands load the library so the API can be used.

|  |  |
| --- | --- |
| ***TCL***  ***puts "Load ixNetwork Tcl API package"***  ***package req IxTclNetwork*** | ***Python***  ***import os***  ***import sys***  ***import time***  ***ixNetPath = ixNetPath = r'C:\Program Files (x86)\Ixia\IxNetwork\8.40-EA\API\Python'***  ***sys.path.append(ixNetPath)***  ***import IxNetwork*** |

## 5.2 [Connect to the Chassis](#ref_TOC5b):

Assign chassis/client/ixNetwork server port/ chassis port HW port information to the variables. These variables are used later for connecting to the chassis.

**Note:** Edit these variable values to match your setup

|  |  |
| --- | --- |
| ***TCL***  ***namespace eval ::ixia {***  ***set ixNServer 10.XXX.XXX.XXX***  ***set ixNPort 8009***  ***}***  ***puts "Connect to IxNetwork Server"***  ***ixNet connect $::ixia::ixNServer -port $::ixia::ixNPort -version 8.40***    ***puts "Create a new config"***  ***ixNet exec newConfig***  ***TCL***  ***puts “Adding chassis and edit hostname ip as your chassis ip”***  ***set availableHardware [ixNet getL [ ixNet getRoot ] availableHardware]***  ***ixNet add $availableHardware "chassis" -hostname 192.16X.XXX.XXX***  ***ixNet commit*** | ***Python***  ***ixNServer = '* *10.XXX.XXX.XXX***  ***ixNPort = '8009'***  ***# get IxNet class***  ***ixNet = IxNetwork.IxNet()***  ***print("Connect to IxNetwork Server")***  ***ixNet.connect(ixNServer, '-port', ixNPort, '-version', '8.40')***  ***print("Create a new config ")***  ***ixNet.execute('newConfig')***  ***Python***  ***print(“Adding chassis and edit hostname ip as your chassis ip”***  ***root = ixNet.getRoot()***  ***chassisObj1 = ixNet.add(root + '/availableHardware’, 'chassis')***  ***ixNet.setAttribute(chassisObj1, '-hostname', ’ 192.16X.XXX.XXX’ )***  ***ixNet.commit()*** |

\*Color coding: Commands - Handle - Child - Attribute -Exec Commands. Refer Table 3.1 for the description of the commands used in the script.

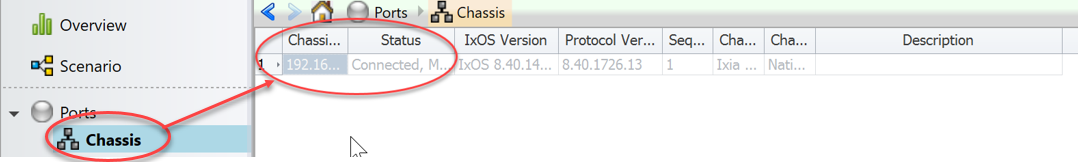


Figure 5.2.1: Chassis got Connected

## 5.3 [Protocol Configuration](#ref_TOC5c):

The follow section is intended to give you the basic idea about the assigning of ports, basic configurations and the configurations required for the protocol.

|  |  |
| --- | --- |
| ***TCL***  ***puts "Adding 2 vports"***  ***ixNet add [ixNet getRoot] vport***  ***ixNet add [ixNet getRoot] vport***  ***ixNet commit***  ***set vPorts [ixNet getList [ixNet getRoot] vport]***  ***set vportTx [lindex $vPorts 0]***  ***set vportRx [lindex $vPorts 1]***  ***puts “get the connected chassis card and its ports”***  ***set chassis [ ixNet getL $availableHardware chassis ]***  ***set card [ ixNet getL $chassis card ]***  ***set ports [ ixNet getL $card port]***  ***puts "Assigning the virtual ports to the physical chassis ports"***  ***ixNet setA $vportTx -connectedTo [lindex $ports 0]***  ***ixNet setA $vportRx -connectedTo [lindex $ports 1]***  ***ixNet commit*** | ***Python***  ***print(“Adding 2 vports")***  ***vport1 = ixNet.add(root, 'vport')***  ***vport1 = ixNet.remapIds(vport1)[0]***  ***vport2 = ixNet.add(root, 'vport')***  ***vport2 = ixNet.remapIds(vport2)[0]***  ***ixNet.commit()***  ***vportTx = ixNet.getList(root, 'vport')[0]***  ***vportRx = ixNet.getList(root, 'vport')[1]***  ***print(“get the connected chassis card and its ports”)***  ***card = ixNet.getList(chassisObj1,'card')***  ***ports = ixNet.getList(card[0],'port')***  ***print(“Assigning the virtual ports to the physical chassis ports"***  ***ixNet.setMultiAttribute(vportTx, '-connectedTo', ports[0], '-rxMode', 'captureAndMeasure', '-name', 'Ethernet - 001')***  ***ixNet.setMultiAttribute(vportRx, '-connectedTo', ports[1], '-rxMode', 'captureAndMeasure', '-name', 'Ethernet - 002')***  ***ixNet.commit()*** |

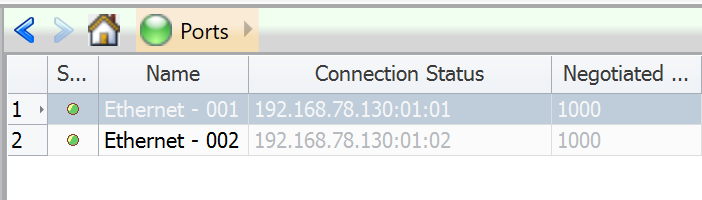


Figure 5.3.1: Showing Chassis Ports got Added and are Up

|  |  |
| --- | --- |
| ***TCL***  ***puts "Adding 2 topologies"***  ***ixNet add [ixNet getRoot] topology -vports $vportTx***  ***ixNet add [ixNet getRoot] topology -vports $vportRx***  ***ixNet commit*** | ***Python***  ***print("adding topologies")***  ***ixNet.add(root, 'topology', '-vports', vportTx)***  ***ixNet.add(root, 'topology', '-vports', vportRx)***  ***ixNet.commit()*** |

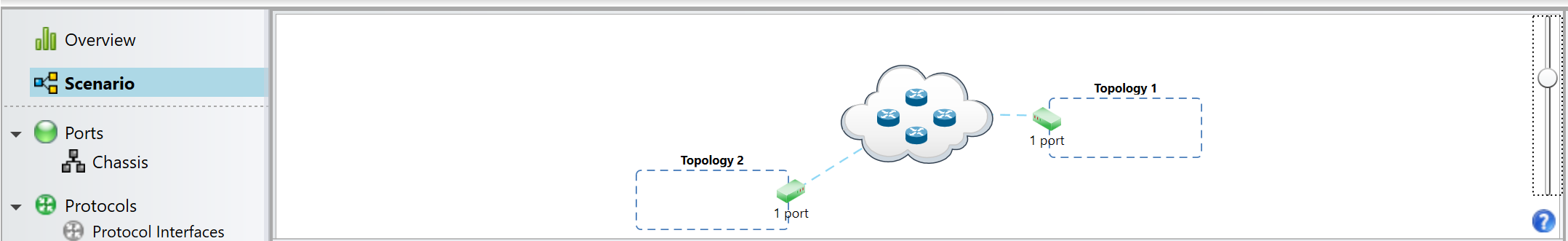
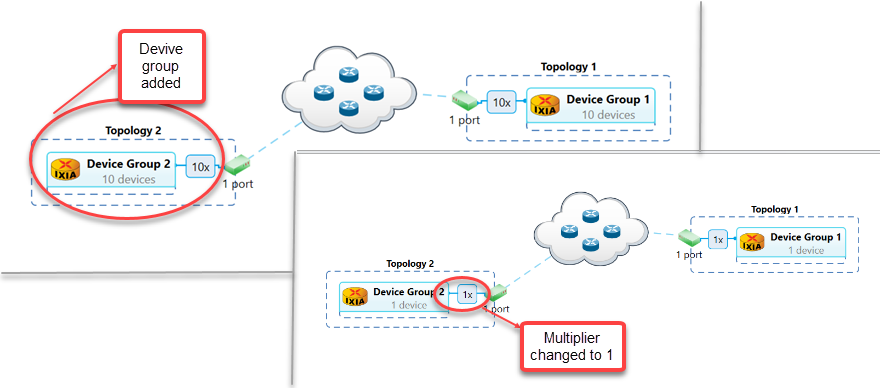


Figure 4.3.2: Showing Topologies are Added

|  |  |
| --- | --- |
| ***TCL***  ***set topologies [ixNet getList [ixNet getRoot] topology]***  ***set topo1 [lindex $topologies 0]***  ***set topo2 [lindex $topologies 1]*** | ***Python***  ***topologies = ixNet.getList(ixNet.getRoot(), 'topology')***  ***topo1 = topologies[0]***  ***topo2 = topologies[1]*** |

|  |  |
| --- | --- |
| ***TCL***  ***puts "Adding 2 device groups"***  ***ixNet add $topo1 deviceGroup***  ***ixNet add $topo2 deviceGroup***  ***ixNet commit*** | ***Python***  ***print "Adding 2 device groups"***  ***ixNet.add(topo1, 'deviceGroup')***  ***ixNet.add(topo2, 'deviceGroup')***  ***ixNet.commit()*** |

|  |  |
| --- | --- |
| ***TCL***  ***set t1devices [ixNet getList $topo1 deviceGroup]***  ***set t2devices [ixNet getList $topo2 deviceGroup]***  ***set t1dev1 [lindex $t1devices 0]***  ***set t2dev1 [lindex $t2devices 0]***  ***puts "Configuring the multipliers (number of sessions)"***  ***ixNet setAttr $t1dev1 -multiplier 1***  ***ixNet setAttr $t2dev1 -multiplier 1***  ***ixNet commit*** | ***Python***  ***t1devices = ixNet.getList(topo1, 'deviceGroup')***  ***t2devices = ixNet.getList(topo2, 'deviceGroup')***  ***t1dev1 = t1devices[0]***  ***t2dev1 = t2devices[0]***  ***print("Configuring the multipliers (number of sessions)")***  ***ixNet.setAttribute(t1dev1, '-multiplier', '1')***  ***ixNet.setAttribute(t2dev1, '-multiplier', '1')***  ***ixNet.commit()*** |

Figure 5.3.3: Showing Device Group is Added and Multiplier Value Changed

|  |  |
| --- | --- |
| ***TCL***  ***puts "Adding ethernet/mac endpoints"***  ***ixNet add $t1dev1 ethernet***  ***ixNet add $t2dev1 ethernet***  ***ixNet commit***  ***set mac1 [ixNet getList $t1dev1 ethernet]***  ***set mac2 [ixNet getList $t2dev1 ethernet]***  ***puts "Configuring the mac addresses"***  ***ixNet setMultiAttr [ixNet getAttr $mac1 -mac]/counter\***  ***-direction increment \***  ***-start {18:03:73:C7:6C:B1} \***  ***-step {00:00:00:00:00:01}***  ***ixNet setAttr [ixNet getAttr $mac2 -mac]/singleValue\***  ***-value {18:03:73:C7:6C:01}***  ***ixNet commit*** | ***Python***  ***print("Adding ethernet/mac endpoints")***  ***ixNet.add(t1dev1, 'ethernet')***  ***ixNet.add(t2dev1, 'ethernet')***  ***ixNet.commit()***  ***mac1 = ixNet.getList(t1dev1, 'ethernet')[0]***  ***mac2 = ixNet.getList(t2dev1, 'ethernet')[0]***  ***print(“Configuring the mac addresses”)***  ***ixNet.setMultiAttribute(ixNet.getAttribute(mac1, '-mac') + '/counter',***  ***'-direction', 'increment',***  ***'-start', '18:03:73:C7:6C:B1',***  ***'-step', '00:00:00:00:00:01')***  ***ixNet.setAttribute(ixNet.getAttribute(mac2, '-mac') + '/singleValue',***  ***'-value', '18:03:73:C7:6C:01')***  ***ixNet.commit()*** |

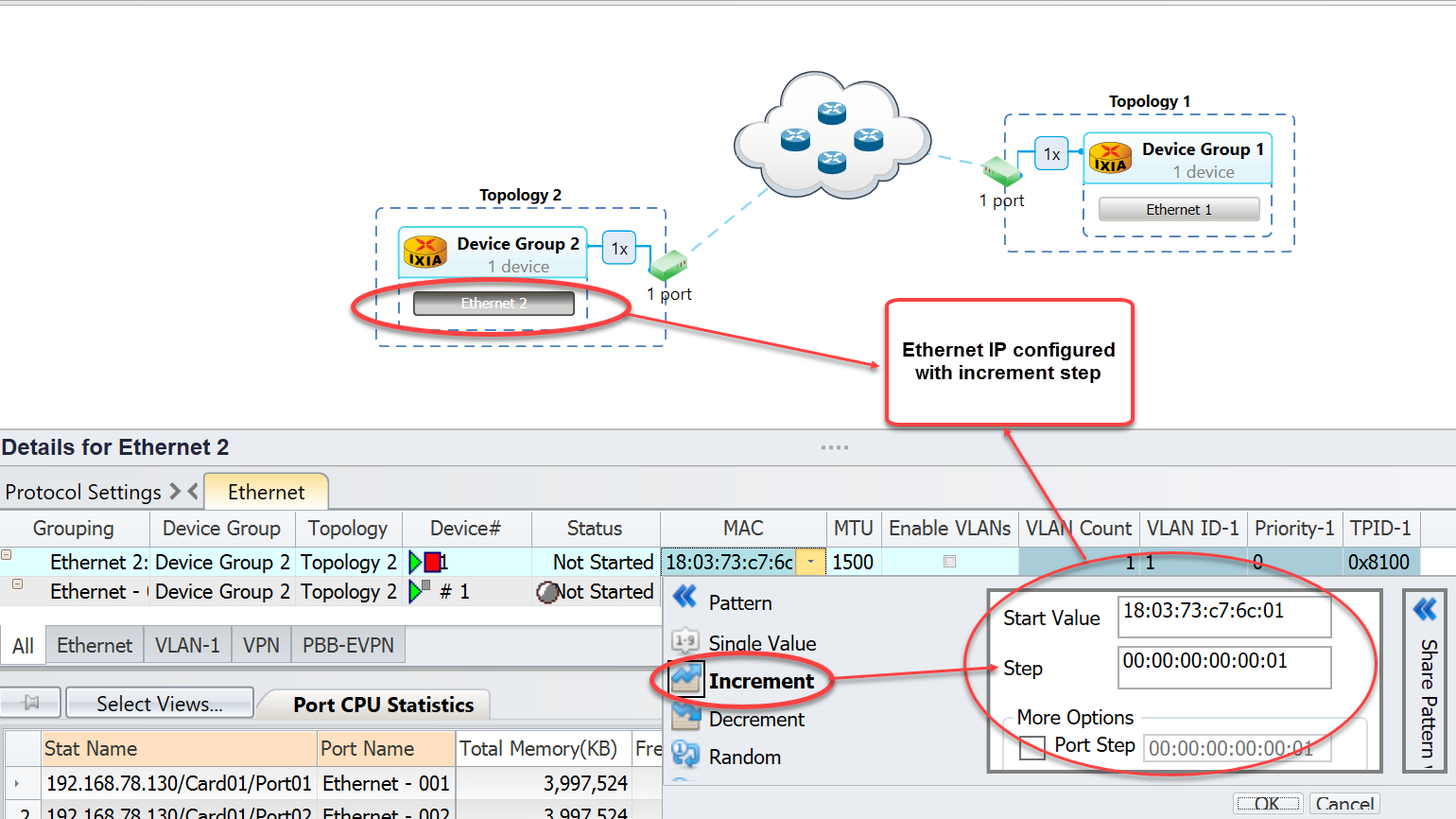


Figure 5.3.4: Showing Mac Address is Configured

|  |  |
| --- | --- |
| ***TCL***  ***puts "Add ipv6"***  ***ixNet add $mac1 ipv6***  ***ixNet add $mac2 ipv6***  ***ixNet commit***  ***set ip1 [ixNet getList $mac1 ipv6]***  ***set ip2 [ixNet getList $mac2 ipv6]***  ***set mvAdd1 [ixNet getAttr $ip1 -address]***  ***TCL***  ***set mvAdd2 [ixNet getAttr $ip2 -address]***  ***set mvGw1 [ixNet getAttr $ip1 -gatewayIp]***  ***set mvGw2 [ixNet getAttr $ip2 -gatewayIp]***  ***puts "configuring ipv6 addresses"***  ***ixNet setAttr $mvAdd1/singleValue -value "11:0:0:0:0:0:0:1"***  ***ixNet setAttr $mvAdd2/singleValue -value "11:0:0:0:0:0:0:2"***  ***ixNet setAttr $mvGw1/singleValue -value "11:0:0:0:0:0:0:2"***  ***ixNet setAttr $mvGw2/singleValue -value "11:0:0:0:0:0:0:1"***  ***ixNet setAttr [ixNet getAttr $ip1 -prefix]/singleValue -value 64***  ***ixNet setAttr [ixNet getAttr $ip2 -prefix]/singleValue -value 64***  ***puts "Adding BGP+ over IPv6 stack"***  ***ixNet add $ip1 bgpIpv6Peer***  ***ixNet add $ip2 bgpIpv6Peer***  ***ixNet commit***  ***set bgp1 [ixNet getList $ip1 bgpIpv6Peer]***  ***set bgp2 [ixNet getList $ip2 bgpIpv6Peer]***  ***puts "Renaming the topologies and the device groups"***  ***ixNet setAttr $topo1 -name "BGP+ Topology 1"***  ***ixNet setAttr $topo2 -name "BGP+ Topology 2"***  ***ixNet setAttr $t1dev1 -name "BGP+ Topology 1 Router"***  ***ixNet setAttr $t2dev1 -name "BGP+ Topology 2 Router"***  ***ixNet commit*** | ***Python***  ***print("Add ipv6")***  ***ixNet.add(mac1, 'ipv6')***  ***ixNet.add(mac2, 'ipv6')***  ***ixNet.commit()***  ***ip1 = ixNet.getList(mac1, 'ipv6')[0]***  ***ip2 = ixNet.getList(mac2, 'ipv6')[0]***  ***mvAdd1 = ixNet.getAttribute(ip1, '-address')***  ***Python***  ***mvAdd2 = ixNet.getAttribute(ip2, '-address')***  ***mvGw1 = ixNet.getAttribute(ip1, '-gatewayIp')***  ***mvGw2 = ixNet.getAttribute(ip2, '-gatewayIp')***  ***print("configuring ipv6 addresses")***  ***ixNet.setAttribute(mvAdd1 + '/singleValue', '-value', '11:0:0:0:0:0:0:1')***  ***ixNet.setAttribute(mvAdd2 + '/singleValue', '-value', '11:0:0:0:0:0:0:2')***  ***ixNet.setAttribute(mvGw1 + '/singleValue', '-value', '11:0:0:0:0:0:0:2')***  ***ixNet.setAttribute(mvGw2 + '/singleValue', '-value', '11:0:0:0:0:0:0:1')***  ***ixNet.setAttribute(ixNet.getAttribute(ip1, '-prefix') + '/singleValue', '-value', '64')***  ***ixNet.setAttribute(ixNet.getAttribute(ip2, '-prefix') + '/singleValue', '-value', '64')***  ***print("Adding BGP+ over IPv6 stacks")***  ***ixNet.add(ip1, 'bgpIpv6Peer')***  ***ixNet.add(ip2, 'bgpIpv6Peer')***  ***ixNet.commit()***  ***bgp1 = ixNet.getList(ip1, 'bgpIpv6Peer')[0]***  ***bgp2 = ixNet.getList(ip2, 'bgpIpv6Peer')[0]***  ***print("Renaming the topologies and the device groups")***  ***ixNet.setAttribute(topo1, '-name', 'BGP+ Topology 1')***  ***ixNet.setAttribute(topo2, '-name', 'BGP+ Topology 2')***  ***ixNet.setAttribute(t1dev1, '-name', 'BGP+ Topology 1 Router')***  ***ixNet.setAttribute(t2dev1, '-name', 'BGP+ Topology 2 Router')***  ***ixNet.commit()*** |

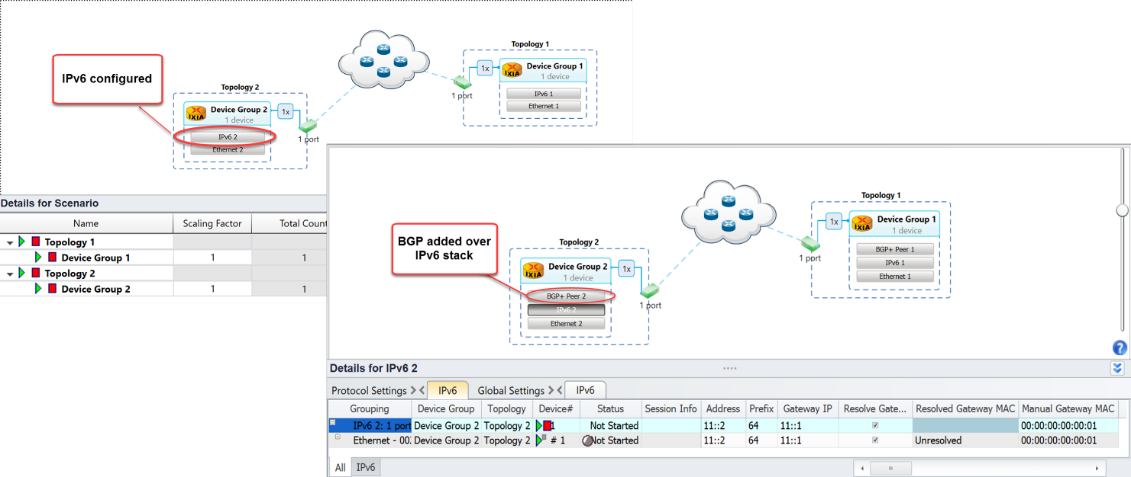


Figure 5.3.5: Showing IPv6 Configuration and BGP Added over IPv6 Stack

|  |  |
| --- | --- |
| ***TCL***  ***puts "Setting IPs in BGP+ DUT IP tab"***  ***ixNet setAttr [ixNet getAttr $bgp1 -dutIp]/singleValue -value "11:0:0:0:0:0:0:2"***  ***ixNet setAttr [ixNet getAttr $bgp2 -dutIp]/singleValue -value "11:0:0:0:0:0:0:1"***  ***ixNet commit*** | ***Python***  ***print("Setting IPs in BGP+ DUT IP tab")***  ***ixNet.setAttribute(ixNet.getAttribute(bgp1, '-dutIp') + '/singleValue', '-value', '11:0:0:0:0:0:0:2')***  ***ixNet.setAttribute(ixNet.getAttribute(bgp2, '-dutIp') + '/singleValue', '-value', '11:0:0:0:0:0:0:1')***  ***ixNet.commit()*** |

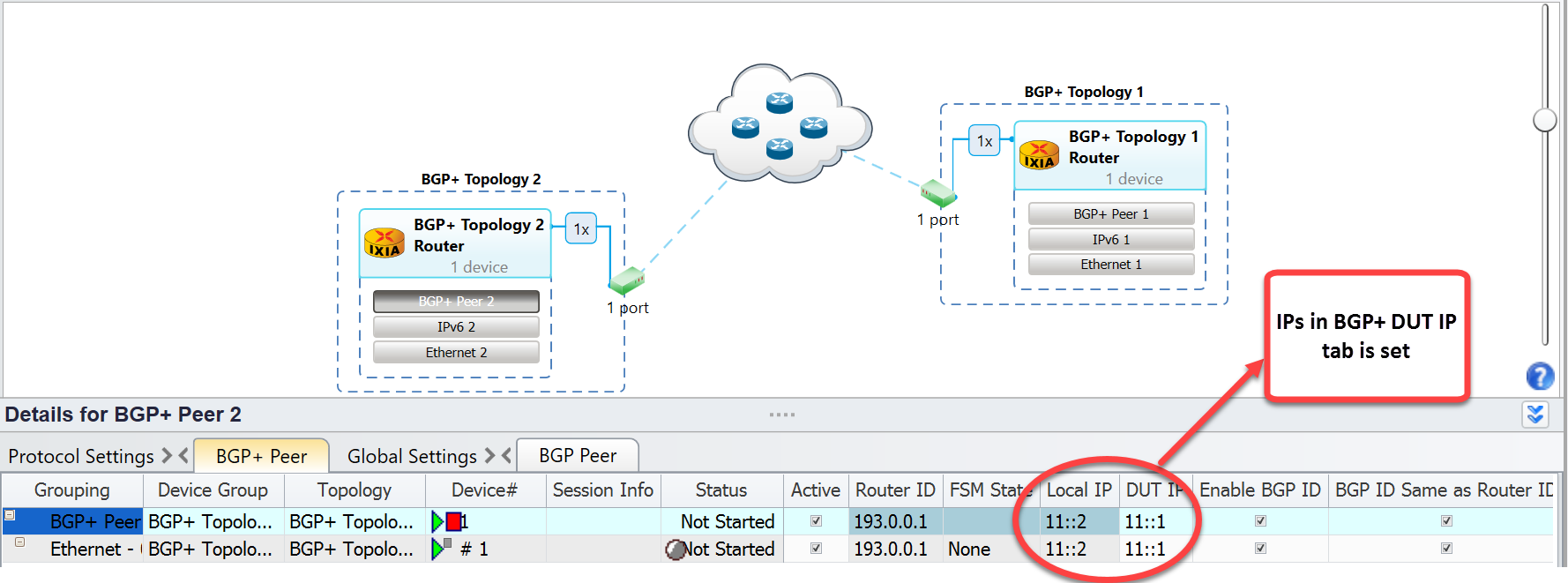


Figure 5.3.6: Showing IPs is Set

|  |  |
| --- | --- |
| ***TCL***  ***puts "Adding the NetworkGroup with Routers at back of it"***  ***ixNet exec createDefaultStack $t1devices ipv6PrefixPools***  ***ixNet exec createDefaultStack $t2devices ipv6PrefixPools***  ***set networkGroup1 [lindex [ixNet getList $t1devices networkGroup] 0]***  ***set networkGroup2 [lindex [ixNet getList $t2devices networkGroup] 0]***  ***ixNet setAttr $networkGroup1 -name "BGP+\_1\_Network\_Group1"***  ***ixNet setAttr $networkGroup2 -name "BGP+\_2\_Network\_Group1"***  ***ixNet commit*** | ***Python***  ***print("Adding the NetworkGroup with Routers at back of it")***  ***ixNet.execute('createDefaultStack', t1devices, 'ipv6PrefixPools')***  ***ixNet.execute('createDefaultStack', t2devices, 'ipv6PrefixPools')***  ***networkGroup1 = ixNet.getList(t1dev1, 'networkGroup')[0]***  ***networkGroup2 = ixNet.getList(t2dev1, 'networkGroup')[0]***  ***ixNet.setAttribute(networkGroup1, '-name', 'BGP+\_1\_Network\_Group1')***  ***ixNet.setAttribute(networkGroup2, '-name', 'BGP+\_2\_Network\_Group1')***  ***ixNet.commit()*** |

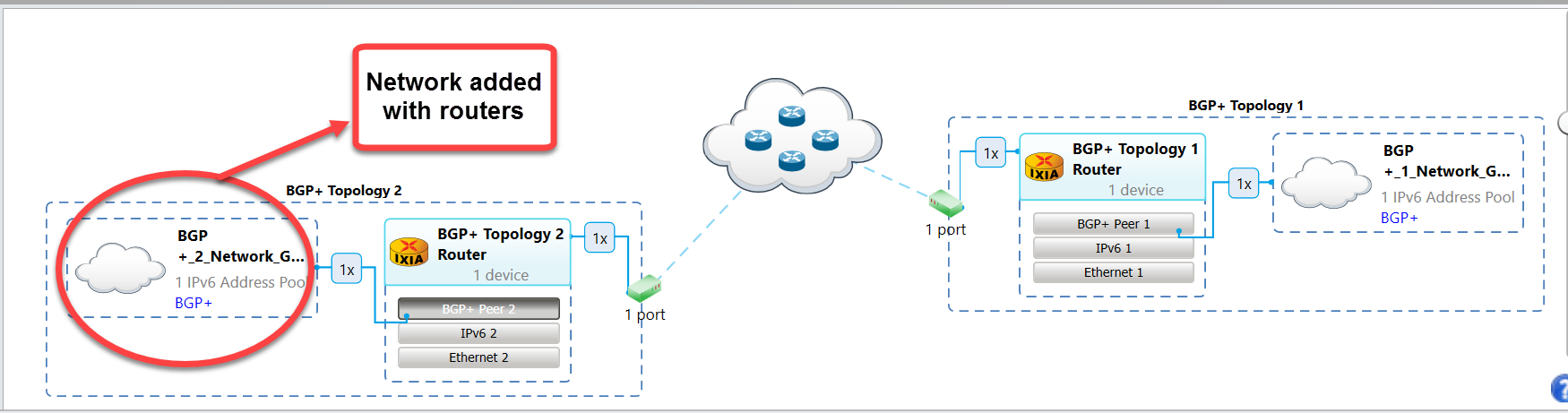


Figure 5.3.7: Showing NetworkGroup with Routers Added at Back of it

####################### add ipv6 loopback1 for applib traffic ###########################

|  |  |
| --- | --- |
| ***TCL***  ***set chainedDg1 [ixNet add $networkGroup1 deviceGroup]***  ***ixNet setMultiAttribute $chainedDg1\***  ***-multiplier 1\***  ***-name {Device Group 4}***  ***ixNet commit***  ***set chainedDg1 [lindex [ixNet remapIds $chainedDg1] 0]***  ***set loopback1 [ixNet add $chainedDg1 "ipv6Loopback"]***  ***ixNet setMultiAttribute $loopback1\***  ***-stackedLayers [list]\***  ***-name {IPv6 Loopback 2}***  ***ixNet commit***  ***set addressSet1 [ixNet getAttribute $loopback1 -address]***  ***ixNet setMultiAttribute $addressSet1\***  ***-clearOverlays false\***  ***-pattern counter***  ***ixNet commit***  ***set addressSet1 [ixNet add $addressSet1 "counter"]***  ***ixNet setMultiAttribute $addressSet1\***  ***-step 0:0:0:0:0:0:0:1\***  ***-start 3000:0:1:1:0:0:0:0\***  ***-direction increment***  ***ixNet commit*** | ***Python***  ***chainedDg1 = ixNet.add(networkGroup1, 'deviceGroup')***  ***ixNet.setMultiAttribute(chainedDg1, '-multiplier', '1', '-name', 'Device Group 4')***  ***ixNet.commit()***  ***chainedDg1 = ixNet.remapIds(chainedDg1)[0]***  ***loopback1 = ixNet.add(chainedDg1, 'ipv6Loopback')***  ***ixNet.setMultiAttribute(loopback1, '-stackedLayers', [], '-name', 'IPv6 Loopback 2')***  ***ixNet.commit()***  ***addressSet1 = ixNet.getAttribute(loopback1, '-address')***  ***ixNet.setMultiAttribute(addressSet1, '-clearOverlays', 'false', '-pattern', 'counter')***  ***ixNet.commit()***  ***addressSet1 = ixNet.add(addressSet1, 'counter')***  ***ixNet.setMultiAttribute(addressSet1, '-step', '0:0:0:0:0:0:0:1', '-start', '3000:0:1:1:0:0:0:0', '-direction', 'increment')***  ***ixNet.commit()*** |

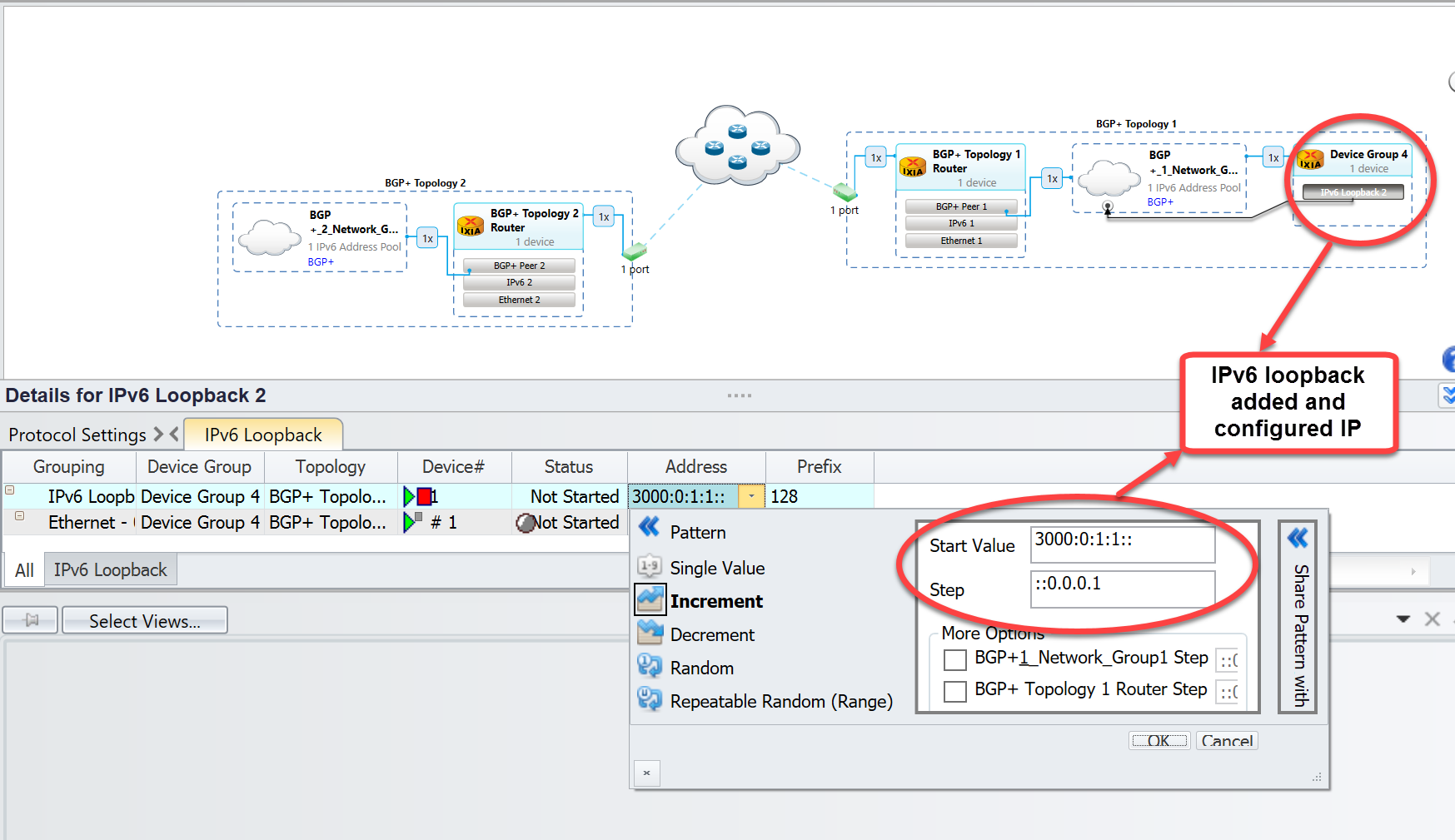


Figure 5.3.8: Showing IPv4 Loopback Added and IP is Configured

######################### add ipv6 loopback2 for applib traffic ##########################

|  |  |
| --- | --- |
| ***TCL***  ***set chainedDg2 [ixNet add $networkGroup2 deviceGroup]***  ***ixNet setMultiAttribute $chainedDg2\***  ***-multiplier 1\***  ***-name {Device Group 3}***  ***ixNet commit***  ***set chainedDg2 [lindex [ixNet remapIds $chainedDg2] 0]***  ***set loopback2 [ixNet add $chainedDg2 "ipv6Loopback"]***  ***ixNet setMultiAttribute $loopback2\***  ***-stackedLayers [list]\***  ***-name {IPv6 Loopback 1}***  ***ixNet commit***  ***set addressSet2 [ixNet getAttribute $loopback2 -address]***  ***ixNet setMultiAttribute $addressSet2\***  ***-clearOverlays false\***  ***-pattern counter***  ***ixNet commit***  ***set addressSet2 [ixNet add $addressSet2 "counter"]***  ***ixNet setMultiAttribute $addressSet2\***  ***-step 0:0:0:0:0:0:0:1\***  ***-start 3000:1:1:1:0:0:0:0\***  ***-direction increment***  ***ixNet commit*** | ***Python***  ***chainedDg2 = ixNet.add(networkGroup2, 'deviceGroup')***  ***ixNet.setMultiAttribute(chainedDg2, '-multiplier', '1', '-name', 'Device Group 3')***  ***ixNet.commit()***  ***chainedDg2 = ixNet.remapIds(chainedDg2)[0]***  ***loopback2 = ixNet.add(chainedDg2, 'ipv6Loopback')***  ***ixNet.setMultiAttribute(loopback2, '-stackedLayers', [], '-name', 'IPv6 Loopback 1')***  ***ixNet.commit()***  ***addressSet2 = ixNet.getAttribute(loopback2, '-address')***  ***ixNet.setMultiAttribute(addressSet2, '-clearOverlays', 'false', '-pattern', 'counter')***  ***ixNet.commit()***  ***addressSet2 = ixNet.add(addressSet2, 'counter')***  ***ixNet.setMultiAttribute(addressSet2, '-step', '0:0:0:0:0:0:0:1', '-start', '3000:1:1:1:0:0:0:0', '-direction', 'increment')***  ***ixNet.commit()*** |

## 5.4 [Start Protocol and Check Statistics](#ref_TOC5d):

The follow section shows how to start the protocols and checking the statistics after protocol has started running.

|  |  |
| --- | --- |
| ***TCL***  **set timer 45000**  ***puts "Starting protocols and waiting for 45 seconds for protocols to come up"***  ***ixNet exec startAllProtocols***  ***after $timer***  ***puts "Verifying all the stats\n"***  ***set viewPage {::ixNet::OBJ-/statistics/view:"Protocols Summary"/page}***  ***set statcap [ixNet getAttr $viewPage -columnCaptions]***  ***foreach statValList [ixNet getAttr $viewPage -rowValues] {***  ***foreach statVal $statValList {***  ***puts "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"***  ***set index 0***  ***foreach satIndv $statVal {***  ***puts [format "%\*s:%\*s" -30 [lindex $statcap $index]\***  ***-10 $satIndv]***  ***incr index***  ***}***  ***}***  ***}*** | ***Python***  ***timer = 45***  ***print("Starting protocols and waiting for 45 seconds for protocols to come up")***  ***ixNet.execute('startAllProtocols')***  ***time.sleep(timer)***  ***print ("Fetching all Protocol Summary Stats\n")***  ***viewPage = '::ixNet::OBJ-/statistics/view:"Protocols Summary"/page'***  ***statcap = ixNet.getAttribute(viewPage, '-columnCaptions')***  ***for statValList in ixNet.getAttribute(viewPage, '-rowValues') :***  ***for statVal in statValList :***  ***print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")***  ***index = 0***  ***for satIndv in statVal :***  ***print("%-30s:%s" % (statcap[index], satIndv))***  ***index = index + 1***  ***# end for***  ***# end for***  ***# end for*** |



Figure 5.4.1: Protocol Comes Up and is Running

############Enabling the IPv6 Learned Information on the fly section ###############################

|  |  |
| --- | --- |
| ***TCL***  ***puts "Enabling IPv6 Unicast Learned Information for BGP+ Router"***  ***ixNet setAttr [ixNet getAttr $bgp1 -filterIpV6Unicast]/singleValue -value true***  ***ixNet setAttr [ixNet getAttr $bgp2 -filterIpV6Unicast]/singleValue -value true***  ***ixNet commit***  ***set globals [ixNet getRoot]/globals***  ***set topology $globals/topology***  ***set timer 10000***  ***if {[catch {ixNet exec applyOnTheFly $topology}] == 1} {***  ***puts "error in applying on the fly change"***  ***puts "$::errorInfo"***  ***}***  ***after $timer*** | ***Python***  ***print("Enabling IPv6 Unicast Learned Information for BGP+ Router")***  ***ixNet.setAttribute(ixNet.getAttribute(bgp1, '-filterIpV6Unicast') + '/singleValue', '-value', 'true')***  ***ixNet.setAttribute(ixNet.getAttribute(bgp2, '-filterIpV6Unicast') + '/singleValue', '-value', 'true')***  ***ixNet.commit()***  ***globalObj = ixNet.getRoot() + '/globals'***  ***topology = globalObj + '/topology'***  ***timer = 10***  ***try :***  ***ixNet.execute('applyOnTheFly', topology)***  ***except :***  ***print("error in applying on the fly change")***  ***# end try/expect***  ***time.sleep(timer)*** |

################### Fetching BGP+ learned info after enabling ipv6 learned info ####################

|  |  |
| --- | --- |
| ***TCL***  ***ixNet exec getIPv6LearnedInfo $bgp1 1***  ***set timer 5000***  ***after $timer***  ***set linfo [ixNet getList $bgp1 learnedInfo]***  ***set values [ixNet getAttribute $linfo -values]***  ***puts "BGP+ learned info"***  ***puts "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"***  ***foreach v $values {***  ***puts $v***  ***}***  ***puts "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"*** | ***Python***  ***ixNet.execute('getIPv6LearnedInfo', bgp1, '1')***  ***timer = 5***  ***time.sleep($timer)***  ***linfo = ixNet.getList(bgp1, 'learnedInfo')[0]***  ***values = ixNet.getAttribute(linfo, '-values')***  ***print(“BGP+ learned info")***  ***print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")***  ***for v in values :***  ***print(v)***  ***# end for***  ***print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")*** |

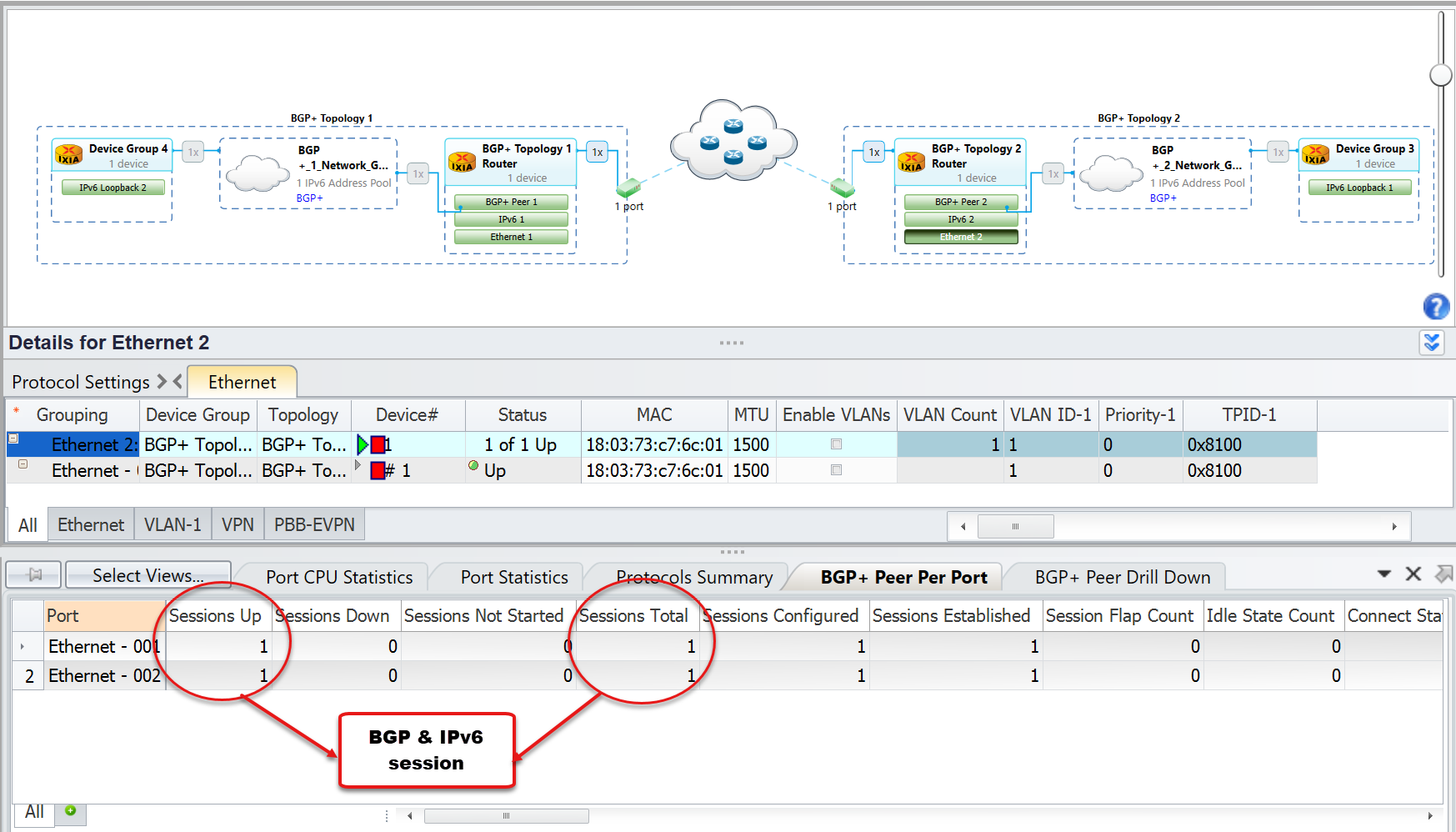


Figure 5.4.2: Statistics on GUI of protocol

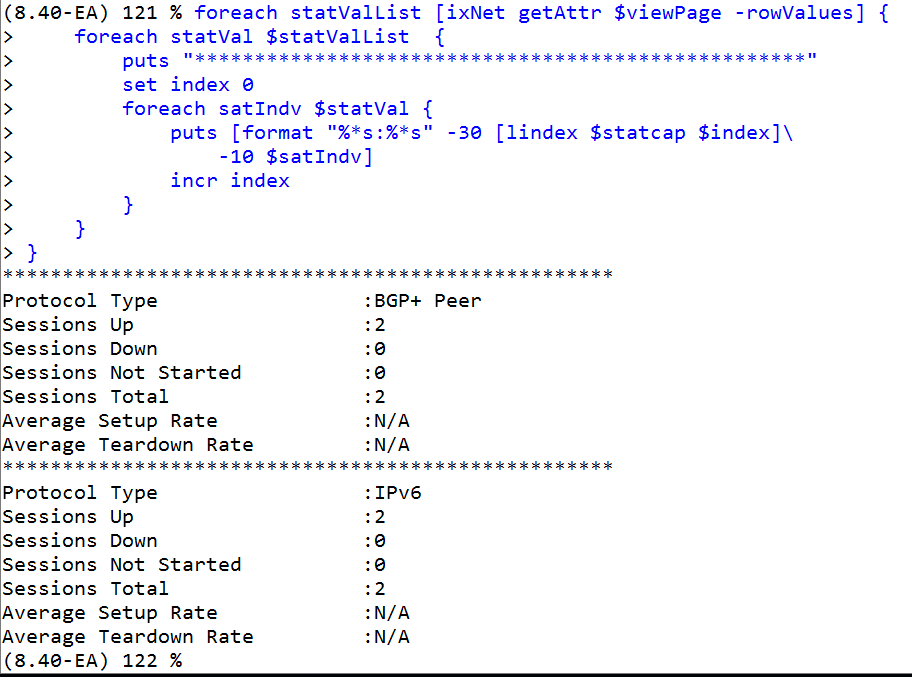


Figure 5.4.3: Statistics Printed on Wish Console is same as seen in Figure 5.4.2

## **5.5** [L2/L3 Traffic Configuration/Apply/Start Section](#ref_TOC5e)**:**

The follow section gives you the idea how to configure traffic and shows how to modify global traffic options.

|  |  |
| --- | --- |
| ***TCL***  ***set trafficItem1 [ixNet add [ixNet getRoot]/traffic "trafficItem"]***  ***ixNet setMultiAttribute $trafficItem1\***  ***-name {Traffic Item 1} \***  ***-roundRobinPacketOrdering false \***  ***-trafficType ipv6***  ***ixNet commit***  ***set trafficItem1 [lindex [ixNet remapIds $trafficItem1] 0]***  ***puts “Adding endpoint set in traffic item”***  ***set endpointSet1 [ixNet add $trafficItem1 "endpointSet"]***  ***set source [list $networkGroup1/ipv6PrefixPools:1]***  ***set destination [list $networkGroup2/ipv6PrefixPools:1]***  ***ixNet setMultiAttribute $endpointSet1\***  ***-name "EndpointSet-1"\***  ***-multicastDestinations [list]\***  ***-scalableSources [list]\***  ***-multicastReceivers [list]\***  ***-scalableDestinations [list]\***  ***-ngpfFilters [list]\***  ***-trafficGroups [list]\***  ***-sources $source\***  ***-destinations $destination\***  ***ixNet commit***  ***ixNet setMultiAttribute $trafficItem1/tracking\***  ***-trackBy [list sourceDestEndpointPair0 trackingenabled0]\***  ***-fieldWidth thirtyTwoBits\***  ***-protocolOffset Root.0\***  ***-values [list]\***  ***ixNet commit*** | ***Python***  ***trafficItem1 = ixNet.add(ixNet.getRoot() + '/traffic', 'trafficItem')***  ***ixNet.setMultiAttribute(trafficItem1, '-name', 'Traffic Item 1',***  ***'-roundRobinPacketOrdering', 'false', '-trafficType', 'ipv6')***  ***ixNet.commit()***  ***trafficItem1 = ixNet.remapIds(trafficItem1)[0]***  ***print(“Adding endpoint set in traffic item”)***  ***endpointSet1 = ixNet.add(trafficItem1, 'endpointSet')***  ***source = [networkGroup1 + '/ipv6PrefixPools:1']***  ***destination = [networkGroup2 + '/ipv6PrefixPools:1']***  ***ixNet.setMultiAttribute(endpointSet1,***  ***'-name', 'EndpointSet-1',***  ***'-multicastDestinations', [],***  ***'-scalableSources', [],***  ***'-multicastReceivers', [],***  ***'-scalableDestinations', [],***  ***'-ngpfFilters', [],***  ***'-trafficGroups', [],***  ***'-sources', source,***  ***'-destinations', destination)***  ***ixNet.commit()***  ***ixNet.setMultiAttribute(trafficItem1 + '/tracking',***  ***'-trackBy', ['sourceDestEndpointPair0', 'trackingenabled0'],***  ***'-fieldWidth', 'thirtyTwoBits',***  ***'-protocolOffset', 'Root.0',***  ***'-values', [])***  ***ixNet.commit()*** |

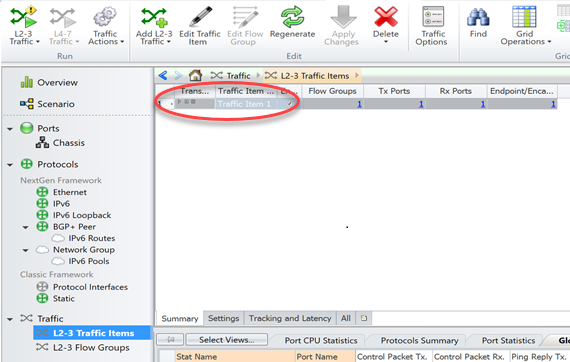


Figure 5.5.1: Showing Traffic Item Added

## 5.6 [Packet Captures](#ref_TOC5f):

IxNetwork's packet capture module has two types of capture: control capture and data capture. Control capture is used to capture protocols packets only and data capture is used to captures both packets and data packets.

|  |  |
| --- | --- |
| ***TCL***  ***puts "Enable Control capture"***  ***set captureObj1 "$vportTx/capture"***  ***ixNet setA $captureObj1 -softwareEnabled true***  ***ixNet commit***  ***puts "Enable Data capture"***  ***set captureObj2 "$vportRx/capture"***  ***ixNet setA $captureObj2 -hardwareEnabled true***  ***ixNet commit*** | ***Python***  ***captureObj1 = ixNet.getList(vportTx,'capture')***  ***ixNet.setAttribute(captureObj1[0], '-softwareEnabled' , 'true')***  ***ixNet.commit()***  ***captureObj2 = ixNet.getList(vportRx,'capture')***  ***ixNet.setAttribute(captureObj2[0], '-hardwareEnabled' , 'true')***  ***ixNet.commit()*** |



Figure 5.6.1: Showing Data and Control Enabled

|  |  |
| --- | --- |
| ***TCL***  ***puts "Starting capture"***  ***set timer 30000***  ***ixNet exec startCapture***  ***after $timer*** | ***Python***  ***print("Starting capture")***  ***timer = 30***  ***ixNet.execute('startCapture')***  ***time.sleep(timer)*** |

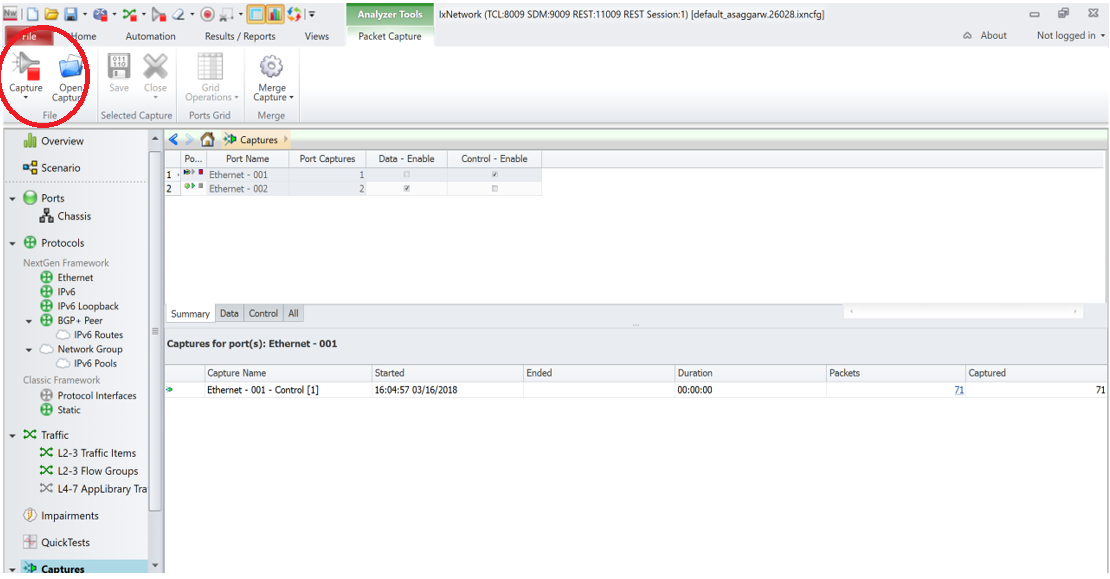


Figure 5.6.2: Showing Capture is Started

## 5.7 [Apply Start Traffic](#ref_TOC5g):

In the follow section it is shown how the traffic is applied and started. Screenshots are shown to show the outcome for the same.

|  |  |
| --- | --- |
| ***TCL***  ***puts "applying traffic"***  ***set timer 5000***  ***ixNet exec apply [ixNet getRoot]/traffic***  ***puts “let the traffic gets applied”***  ***after $timer***  ***puts "starting traffic"***  ***ixNet exec start [ixNet getRoot]/traffic***  ***set timer 60000***  ***puts "let traffic run for 60 seconds"***  ***after $timer*** | ***Python***  ***print ('applying L2/L3 traffic')***  ***timer = 5***  ***ixNet.execute('apply', ixNet.getRoot() + '/traffic')***  ***print(“let the traffic gets applied”)***  ***time.sleep(timer)***  ***print ('starting traffic')***  ***ixNet.execute('start', ixNet.getRoot() + '/traffic')***  ***timer = 60***  ***print("let traffic run for 60 seconds")***  ***time.sleep(timer)*** |

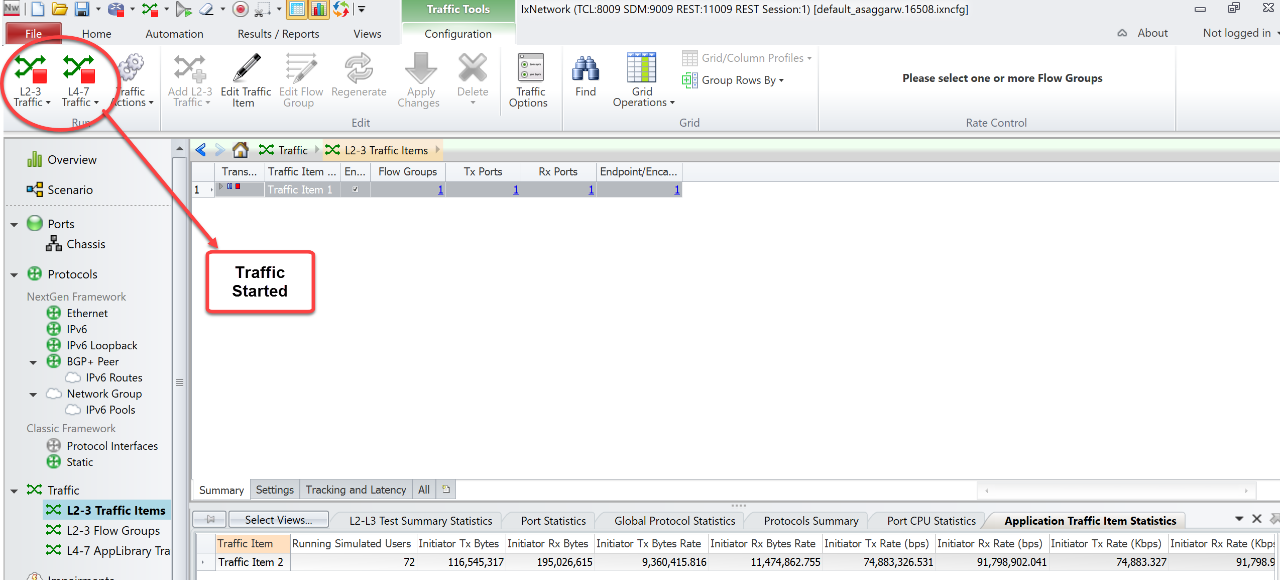


Figure 5.7.1: Traffic is Started

## 5.8 [Retrieving Statistics](#ref_TOC5h):

The follow section shows all the traffic statistics collected when the traffic is running and the packets captured.

|  |  |
| --- | --- |
| ***TCL***  ***puts "Stopping capture"***  ***set timer 30000***  ***ixNet exec stopCapture***  ***after $timer*** | ***Python***  ***print(“Stopping capture")***  ***timer = 30***  ***ixNet.execute('stopCapture')***  ***time.sleep(timer)*** |



Figure 5.8.1: Showing Packets Captured after Stopping Captures

##################### print traffic item statistics #############################

|  |  |
| --- | --- |
| ***TCL***  ***puts "Verifying all the application traffic stats\n"***  ***set viewPage {::ixNet::OBJ-/statistics/view:"Flow Statistics"/page }***  ***set statcap [ixNet getAttr $viewPage -columnCaptions]***  ***foreach statValList [ixNet getAttr $page -rowValues] {***  ***foreach statVal $statValList {***  ***puts "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"***  ***set index 0***  ***foreach satIndv $statVal {***  ***puts [format "%\*s:%\*s" -30 [lindex $statcap $index]\***  ***-10 $satIndv]***  ***incr index***  ***}***  ***}***  ***}*** | ***Python***  ***print ('Verifying all the L2-L3 traffic stats')***  ***viewPage = '::ixNet::OBJ-/statistics/view:"Flow Statistics"/page'***  ***statcap = ixNet.getAttribute(viewPage, '-columnCaptions')***  ***for statValList in ixNet.getAttribute(viewPage, '-rowValues') :***  ***for statVal in statValList :***  ***index = 0***  ***for satIndv in statVal :***  ***print("%-30s:%s" % (statcap[index], satIndv))***  ***index = index + 1***  ***# end for***  ***# end for***  ***# end for*** |

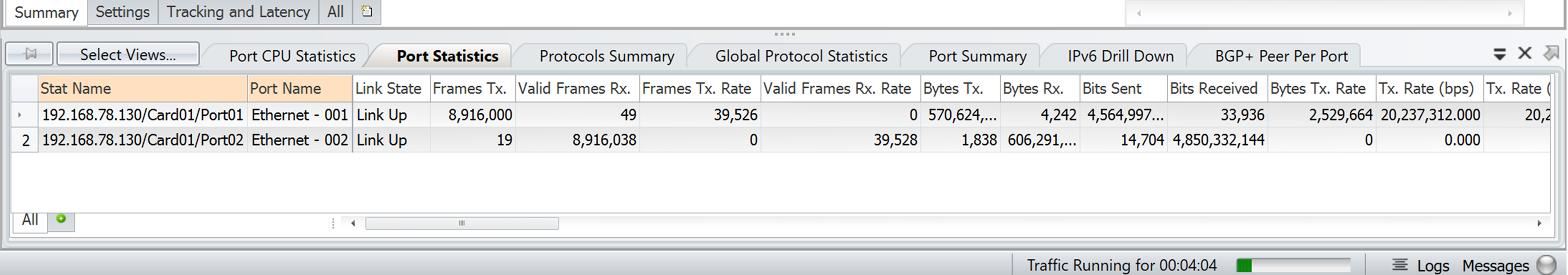


Figure 5.8.2: Showing Traffic Item Statistics

|  |  |
| --- | --- |
| ***TCL***  ***puts "ixNet help ::ixNet::OBJ-/topology/deviceGroup/ethernet"***  ***puts "[ixNet help ::ixNet::OBJ-/topology/deviceGroup/ethernet]"*** | ***Python***  ***print ('ixNet.help(\'::ixNet::OBJ-/topology/deviceGroup/ethernet\')')***  ***print (ixNet.help('::ixNet::OBJ-/topology/deviceGroup/ethernet'))*** |

|  |  |
| --- | --- |
| ***TCL***  ***puts "stopping traffic"***  ***ixNet exec stop [ixNet getRoot]/traffic***  ***set timer 5000***  ***puts “let the traffic stops”***  ***after $timer***  ***puts "Stopping all protocols"***  ***ixNet exec stopAllProtocols***  ***puts “let the protocol stops”***  ***after $timer*** | ***Python***  ***print ('Stopping traffic')***  ***ixNet.execute('stop', ixNet.getRoot() + '/traffic')***  ***timer = 5***  ***print(‘let the traffic stops’)***  ***time.sleep(timer)***  ***print ('Stopping all protocols')***  ***ixNet.execute('stopAllProtocols')***  ***print(“let the protocol stops”)***  ***time.sleep(timer)*** |

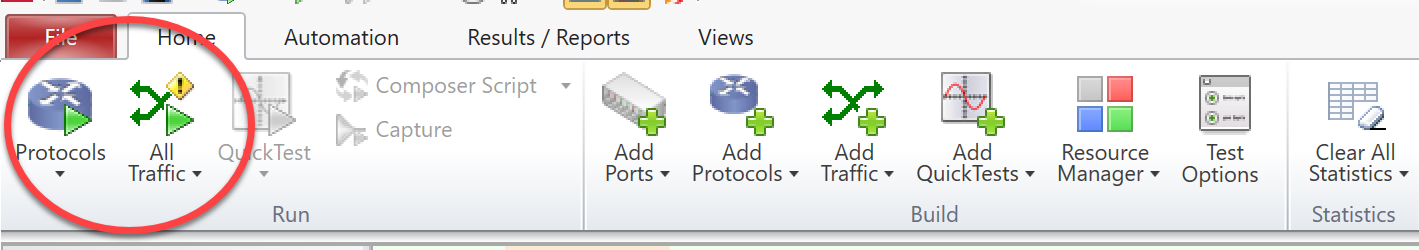
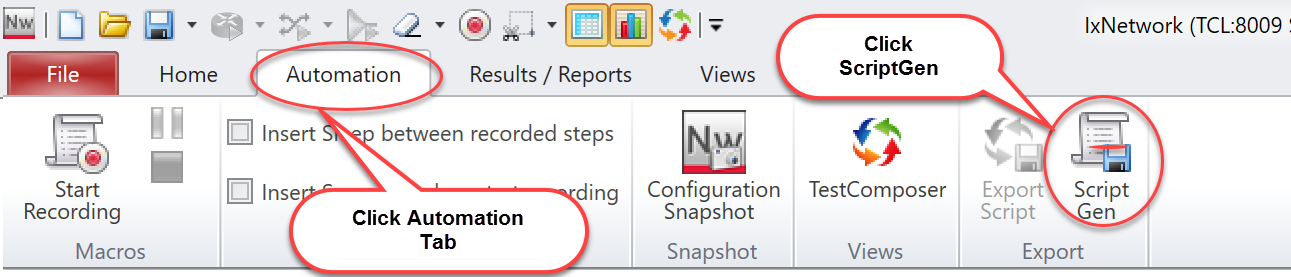


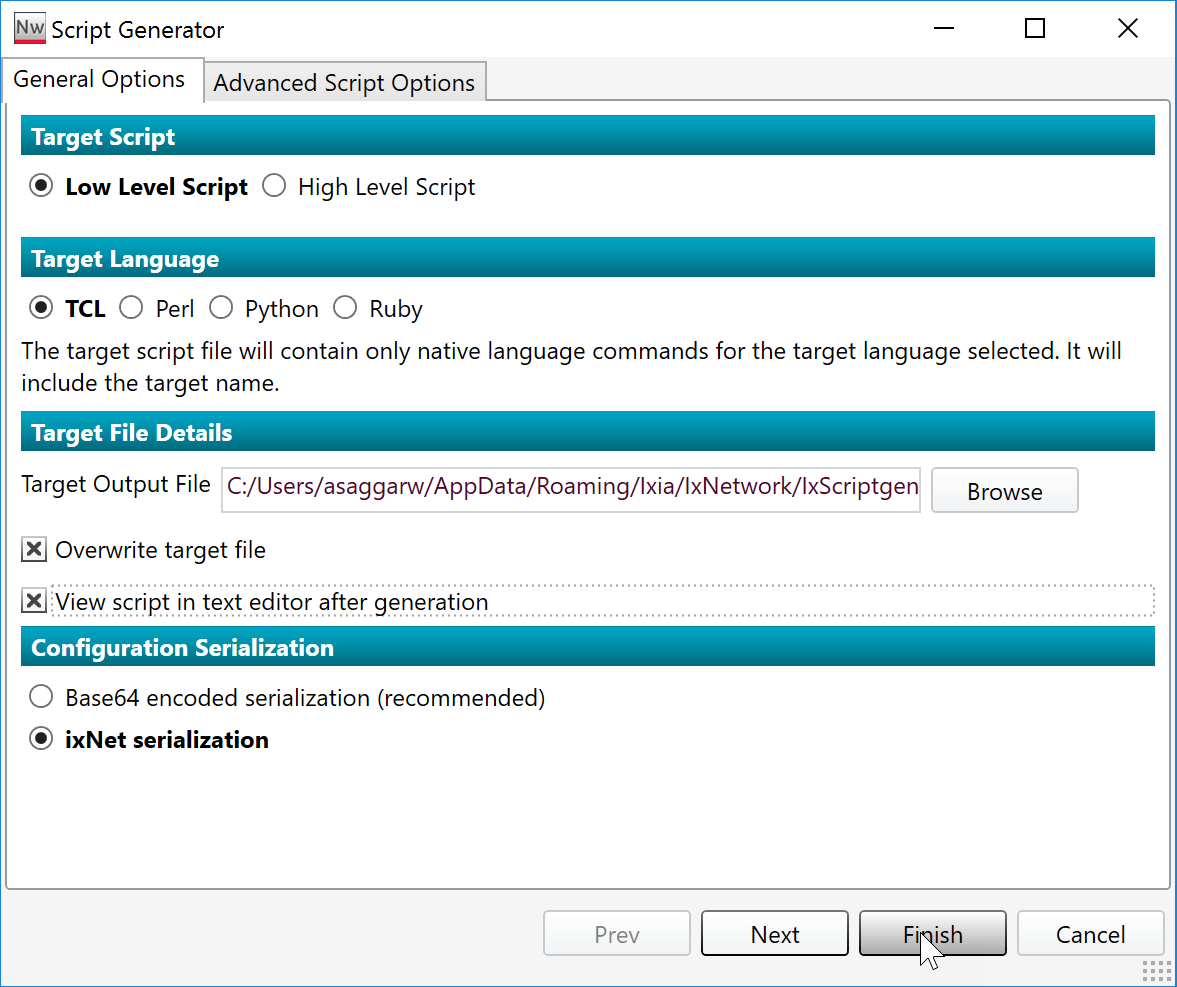
Figure 5.8.3: Showing Protocol and Traffic is Stopped

## 5.9 [Generate Scriptgen](#ref_TOC5i):

On the IxNetwork GUI's toolbar, click Automation, and then click ScriptGen.



Make your selections, and then in the bottom section, click ixNet Serialization. Click Finish.



Serialized to the target script file

as a base64 encoded method.

This is a fast method for all

sizes of configuration.

Serialized to the target script file as ixNet commands.

# 6. [References](#ref_TOC6):

1. Discover the hierarchical structure by using API browser:

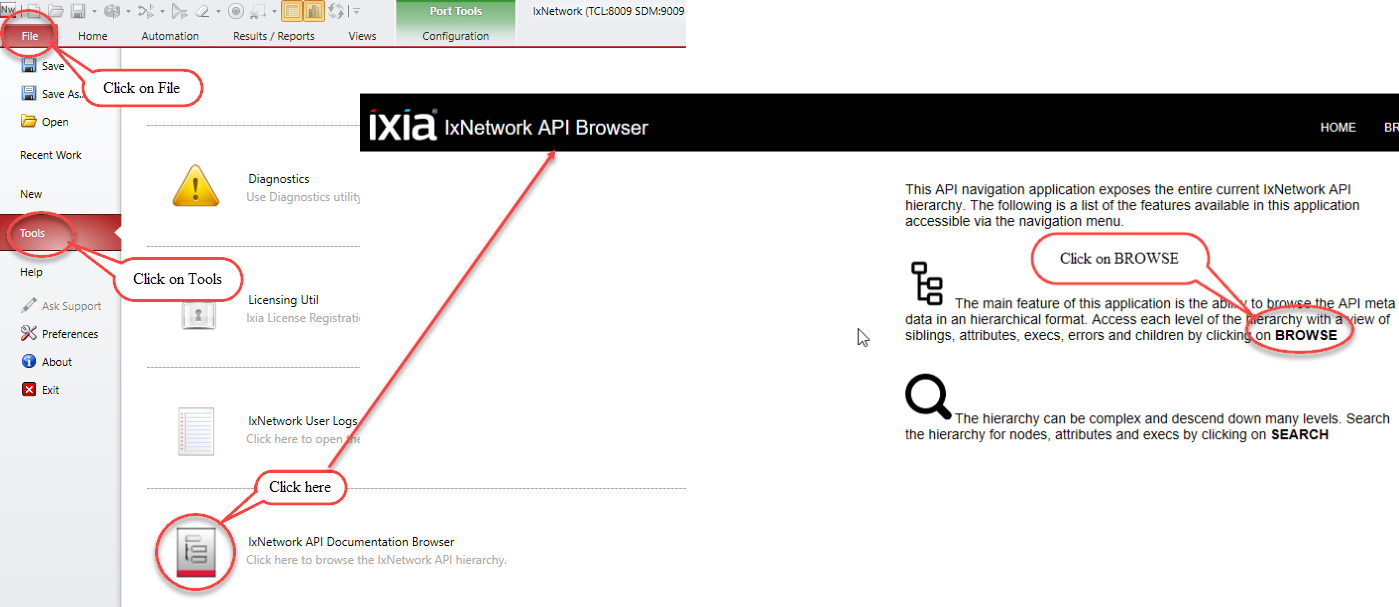


Figure 6.1: Accessing the IxNetwork API Browser

2. For more information, see <https://github.com/OpenIxia/IxNetwork/tree/master/LowLevelApi>.

# 7. [Support](#ref_TOC7):

For more information, visit <https://support.ixiacom.com/>.

For support assistance, contact [support.ix@keysight.com](mailto:support.ix@keysight.com).