

```
//-----\\
\\//-----\\
Get-FENetwork [~] [FightingEntropy(π)][2022.12.0]
-----\\
\\//-----\\
```

Introduction /

Greetings,

This is going to be a rather brief overview of the (function/tool) Get-FENetwork in the upcoming release of [FightingEntropy(π)]. It is not QUITE complete, as there are a couple of minor issues that I've resolved, but the string output is not showing correctly for (2) areas related to NBT Host count.

Other than that...?

This function has been greatly enhanced over all prior versions of the function. There are (55) classes within the function, most of them are simply list classes that contain subclasses and methods so that the controller factory class can produce all of the embedded classes within the utility.

In other words, some of the methods of the output are able to produce the internal classes, either to make debugging easier to do, or for adding additional properties or extensions in the future.

I have covered the code-behind in a (lesson plan + video) that covers the Xaml construction process, as well as parsing classes for (ARP/NBT).

| 03/2021 | A Deep Dive: PowerShell and XAML | <https://youtu.be/NK4NuQrraCI> |

The associated lesson plan (*.pdf) is available in the details of that video description, and it largely covers many of the following classes, albeit to a much lesser extent.

The current version of this utility is meant to be QUICK, THOROUGH, and POWERFUL. Many of these classes reach into CimInstance classes native to [Windows PowerShell]. When they do, the properties are pulled into the parent class, and then they're handed off to the network controller template, where they're sorted into corresponding address families and then each individual IP address for each adapter and address family are handed their own compartment.

This is specifically meant to be extremely thorough and to provide redundancy, and I WILL be implementing these properties to the (GUI/graphical user interface) utility that runs in tandem of...

| 12/02/22 | [https://github.com/mcc85s/FightingEntropy/blob/main/Docs/2022_1202-\(Get-ViperBomb\).pdf](https://github.com/mcc85s/FightingEntropy/blob/main/Docs/2022_1202-(Get-ViperBomb).pdf) |

...as well as the event log viewer utility.

Like I said, this thing is going to be quite complex and have very fine grained control, and room for growth.

We're not building utilities for little girls to have a sleepover and a tea party...

Nah.

We're going way further than that... so, being detail oriented and having this level of control is CRITICAL.

/ Introduction

```
<#
.SYNOPSIS
.DESCRIPTION
.LINK
.NOTES

//=====\\
// Module      : [FightingEntropy()][2022.12.0]
// Date        : 2022-12-21 13:46:41
//=====\\

FileName      : Get-FENetwork.ps1
Solution      : [FightingEntropy()][2022.12.0]
Purpose       : Extensive and elaborate utility meant to obtain and organize:
                [+] hardware vendors
                [+] (ARP/address resolution protocol) information
```

```

    [+] (NBT/NetBEUI table) statistics
    [+] (NetStat/network) statistics
    [+] network interfaces
    [+] network IP addresses
    [+] network routes
    [+] extended WMI/CIM properties
    [+] compartments for each individual address family and type
    [+] extensions that manage (V4/V6) network capabilities
    [+] filtering everything into their own corresponding output types
    [+] allowing for a domain controller to join an Active Directory domain (DCPromo)
    [+] able to be extended with additional features

Author      : Michael C. Cook Sr.
Contact     : @mcc85s
Primary     : @mcc85s
Created     : 2022-12-14
Modified    : 2022-12-21
Demo        : N/A
Version     : 0.0.0 - () - Finalized functional version 1
TODO        : NBT scan remote addresses

```

.Example

```

=====
| 00 | [ ] Vendor [ ] Arp/Nbt [ ] Netstat |
| 01 | [ ] Vendor [ ] Arp/Nbt [X] Netstat |
| 02 | [ ] Vendor [X] Arp/Nbt [ ] Netstat |
| 03 | [ ] Vendor [X] Arp/Nbt [X] Netstat |
| 04 | [X] Vendor [ ] Arp/Nbt [ ] Netstat |
| 05 | [X] Vendor [ ] Arp/Nbt [X] Netstat |
| 06 | [X] Vendor [X] Arp/Nbt [ ] Netstat |
| 07 | [X] Vendor [X] Arp/Nbt [X] Netstat |
=====

#>

Function Get-FENetwork
{
    [CmdLetBinding()]Param([Parameter()][UInt32]$Mode=0)

```

```
Class [Time] /-----\
/-----\
```

```

# // =====
# // | Creates a time object similar to the [System.Diagnostics.Stopwatch] object, but is much simpler. |
# // =====

Class Time
{
    Hidden [Object] $Start
    Time()
    {
        $This.Start = [DateTime]::Now
    }
    [String] ToString()
    {
        Return [Timespan]([DateTime]::Now-$This.Start)
    }
}

```

```
Enum [ModeType] /-----\
/-----\
```

```
Class [Time]
```

```

# // =====
# // | Enum types for FENetwork mode switch |
# // =====

Enum ModeType

```

```

{
    None
    NetstatOnly
    ArpNbtOnly
    ArpNbtNetstat
    VendorOnly
    VendorNetstat
    VendorArpNbt
    All
}

```

```

Class [ModeItem] /

```

```

Enum [ModeType]

```

```

# // =====
# // | Individual FENetwork mode switch |
# // =====

Class ModeItem
{
    [UInt32]      $Index
    [String]      $Type
    [String] $Description
    ModeItem([UInt32]$Index,[String]$Type)
    {
        $This.Index = $Index
        $This.Type = [ModeType]::$Type
    }
    [String] ToString()
    {
        Return $This.Index
    }
}

```

```

Class [ModeList] /

```

```

Class [ModeItem]

```

```

# // =====
# // | List of all FENetwork mode switches |
# // =====

Class ModeList
{
    [String]      $Name
    [UInt32]      $Count
    [Object]      $Output
    [Object] $Selected
    ModeList()
    {
        $This.Name = "ModeList"
        $This.Output = @( )

        ForEach ($Name in [System.Enum]::GetNames([ModeType]))
        {
            $This.Add($Name)
        }

        $This.Selected = $This.Select(0)
    }
    Add([String]$Type)
    {
        $Item = [ModeItem]::New($This.Output.Count,$Type)
        $Item.Description = Switch ($Item.Index)
    }
}

```

```

    {
        0 { "[ ] Vendor [ ] Arp/Nbt [ ] Netstat" }
        1 { "[ ] Vendor [ ] Arp/Nbt [X] Netstat" }
        2 { "[ ] Vendor [X] Arp/Nbt [ ] Netstat" }
        3 { "[ ] Vendor [X] Arp/Nbt [X] Netstat" }
        4 { "[X] Vendor [ ] Arp/Nbt [ ] Netstat" }
        5 { "[X] Vendor [ ] Arp/Nbt [X] Netstat" }
        6 { "[X] Vendor [X] Arp/Nbt [ ] Netstat" }
        7 { "[X] Vendor [X] Arp/Nbt [X] Netstat" }
    }
    $This.Output += $Item
    $This.Count = $This.Output.Count
}
Select([UInt32]$Index)
{
    $This.Selected = $This.Output[$Index]
}
[String] ToString()
{
    Return $This.Selected.Index
}
}

```

```
Class [DNSSuffix]
```

```
Class [ModelList]
```

```

# // =====
# // | Collects DNS Suffix/registration information |
# // =====

Class DNSSuffix
{
    [UInt32] $IsDomain
    [String] $ComputerName
    [String] $Domain
    [String] $NVDomain
    [UInt32] $Sync
    DNSSuffix()
    {
        $This.IsDomain = $This.GetComputerSystem().PartOfDomain
        $Item = $This.GetParameters()
        $This.ComputerName = $Item.HostName
        $This.Domain = @("-", $Item.Domain)[$This.IsDomain]
        $This.NVDomain = @("-", $Item.'NV Domain')[$This.IsDomain]
        $This.Sync = $Item.SyncDomainWithMembership
    }
    [Object] GetParameters()
    {
        Return Get-ItemProperty "HKLM:\System\CurrentControlSet\Services\TCPIP\Parameters"
    }
    [Object] GetComputerSystem()
    {
        Return Get-CimInstance Win32_ComputerSystem
    }
    SetDomain([String]$Domain)
    {
        $This.Domain = $Domain
    }
    SetComputerName([String]$ComputerName)
    {
        $This.ComputerName = $ComputerName
    }
    SetSync()
    {
        If (!$This.IsDomain)
        {
            ForEach ($Item in "Domain", "NV Domain")
            {

```

```

        Set-ItemProperty -Path $This.Path -Name $Item -Value $This.Domain -Verbose
    }
}

Else
{
    Throw "System is part of a domain"
}
}
[String] ToString()
{
    Return $This.Domain
}
}
}

```

```
Class [VendorItem] /
```

```
/ Class [DNSSuffix]
```

```

# // =====
# // | Individual vendor information for the vendor list |
# // =====

Class VendorItem
{
    [String] $Index
    [String] $Hex
    [String] $Name
    VendorItem([UInt32]$Index,[String]$Line)
    {
        $This.Index = $Index
        $This.Hex, $This.Name = $Line -Split "\t"
    }
    [String] ToString()
    {
        Return "<FENetwork.VendorItem>"
    }
}
}

```

```
Class [VendorList] /
```

```
/ Class [VendorItem]
```

```

# // =====
# // | Collects vendor information from the vendor list |
# // =====

Class VendorList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    VendorList()
    {
        $This.Name = "VendorList"
        $Module = Get-FEModule -Mode 1
        $Path = $Module._Control("vendorlist.txt").Fullname

        If (![System.IO.File]::Exists($Path))
        {
            Throw "Unable to locate the vendor list file"
        }

        $File = [System.IO.File]::ReadAllLines($Path)
        $Hash = @{ }
    }
}

```

```

        ForEach ($Line in $File)
        {
            $Hash.Add($Hash.Count,$This.VendorItem($Hash.Count,$Line))
        }

        $This.Output = $Hash[0..($Hash.Count-1)]
        $This.Count = $This.Output.Count
    }
    [Object] VendorItem([UInt32]$Index,[String]$Line)
    {
        Return [VendorItem]::New($Index,$Line)
    }
    [String] Find([String]$Mac)
    {
        $ID      = $Mac -Replace "(-|:)", "" | % Substring 0 6
        $Item     = $This.Output | ? Hex -eq $ID

        If (!$Item)
        {
            $Item = $This.VendorItem(0,"0`-t-")
        }

        Return $Item.Name
    }
    [String] ToString()
    {
        Return "{0} <FENetwork.VendorList>" -f $This.Count
    }
}

```

```
Class [ArpHost]
```

```
Class [VendorList]
```

```

# // =====
# // | Collects/formats the information for an ARP entry |
# // =====

Class ArpHost
{
    [String] $IPAddress
    [String] $Physical
    [String] $Type
    ArpHost([String]$Line)
    {
        $This.IPAddress = [Regex]::Matches($Line,"(\d+\.\.){3}\d+").Value
        $This.Physical  = [Regex]::Matches($Line,"([0-9a-f]{2}\-){5}[0-9a-f]{2}").Value
        $This.Type      = $Line.Substring(46).TrimEnd(" ")
        $This.GetAssociation()
    }
    GetAssociation()
    {
        $Split = $This.IPAddress.ToString().Split(".")
        If ($Split[0] -in 224,239)
        {
            $This.Type = "Multicast"
        }
        If ($Split[0] -eq 255)
        {
            $This.Type = "Broadcast"
        }
        If ($This.Physical -match "(ff\-){5}ff")
        {
            If ($Split[0] -ne 255)
            {
                $This.Type = "HostMax"
            }
        }
        Else

```

```

        {
            $This.Type = "Broadcast"
        }
    }
    If ($This.Type -eq "dynamic")
    {
        $This.Type = "Host"
    }
}
[String] ToString()
{
    Return "<FENetwork.Arphost>"
}
}

```

```
Class [ArpAdapter] /
```

```
/ Class [Arphost]
```

```

# // =====
# // | Collects/Formats information for an adapter in the ARP table |
# // =====

Class ArpAdapter
{
    [UInt32]    $Index
    [String]    $Type
    [String]    $IpAddress
    [Object]    $Host
    ArpAdapter([String]$Line)
    {
        $This.Index    = [Regex]::Matches($Line,"(0x\d+)").Value
        $This.IpAddress = [Regex]::Matches($Line,"(\d+\.\.){3}\d+").Value
        $This.Type      = @("Public","Private")[$This.IpAddress -match 169.254]
        $This.Host      = @( )
    }
    Add([String]$Line)
    {
        $This.Host += [Arphost]::New($Line)
    }
    [String] ToString()
    {
        Return "<FENetwork.ArpAdapter>"
    }
}
}

```

```
Class [ArpList] /
```

```
/ Class [ArpAdapter]
```

```

# // =====
# // | Collects/Formats the entire ARP table |
# // =====

Class ArpList
{
    [String]    $Name
    [UInt32]    $Count
    [Object]    $Output
    ArpList()
    {
        $This.Name = "ArpList"
    }
    Refresh()
}

```

```

{
    $Query      = arp -a
    $This.Output = @( )
    ForEach ($X in 0..($Query.Count-1))
    {
        $Line = $Query[$X]
        Switch -Regex ($Line)
        {
            "^Interface"
            {
                $This.Output += [ArpAdapter]::New($Line)
            }
            "^s{2}\d"
            {
                $This.Output[$This.Output.Count-1].Add($Line)
            }
        }
    }
    $This.Count = $This.Output.Count
}
[String] ToString()
{
    Return "{0}" <FENetwork.ArpList>" -f $This.Count
}
}

```

```
Class [NbtStatReference] /
```

```
/ Class [ArpList]
```

```

# // =====
# // | Object to populate the NBT Reference Table |
# // =====

Class NbtStatReference
{
    [String] $ID
    [String] $Type
    [String] $Service
    NbtStatReference([String]$In)
    {
        $This.ID, $This.Type, $This.Service = $In -Split "/"
        $This.ID = "<${$This.ID}>"
    }
    [String] ToString()
    {
        Return "<FENetwork.NbtStatReference>"
    }
}

```

```
Class [NbtStatHost] /
```

```
/ Class [NbtStatReference]
```

```

# // =====
# // | Information about detected NBT hosts |
# // =====

Class NbtStatHost
{
    [UInt32] $Index
    [String] $Name
    [String] $Id
    [String] $Type
    [String] $Service

```



```

NbtStatHost([UInt32]$Index,[String]$Line)
{
    $Item          = $Line -Split " " | ? Length -gt 0
    $This.Index    = $Index
    $This.Name     = $Item[0]
    $This.Id       = $Item[1]
    $This.Type     = $Item[2]
}
[String] ToString()
{
    Return "<FENetwork.NbtStatHost>"
}
}

```

```
Class NbtStatInterface
```

```
Class NbtStatHost
```

```

# // =====
# // | Information from netstat table |
# // =====

Class NbtStatInterface
{
    [UInt32]      $Index
    [String]      $Type
    [String]      $Name
    [String]      $IpAddress
    [String]      $Node
    [UInt32]      $Count
    [Object]      $Output
    NbtStatInterface([String]$Type,[String]$Line)
    {
        $Line          = $Line.TrimEnd(" ")
        $This.Type      = $Type
        $This.Name      = $Line.TrimEnd(":")
        $This.Output    = @( )
    }
    AddNode([String]$Line,[String]$IpAddress)
    {
        $Split          = $Line -Replace "Scope","`nScope" -Split "`n"
        $This.IpAddress = [Regex]::Matches($Split[0],"(\d+\.\.){3}\d+").Value
        $This.Node      = @($This.IpAddress,$IpAddress)[!$IpAddress]
    }
    AddHost([String]$Line)
    {
        $This.Output    += [NbtStatHost]::New($This.Output.Count,$Line)
        $This.Count     = $This.Output.Count
    }
    [String] ToString()
    {
        Return "<FENetwork.NbtStatInterface>"
    }
}
}

```

```
Class NbtStatList
```

```
Class NbtStatInterface
```

```

# // =====
# // | Collects the local NBT table (will be modified for remote) |
# // =====

Class NbtStatList

```

```

{
    Hidden [Object] $Reference
    [String]      $Name
    [UInt32]      $Count
    [Object]      $Output
    NbtStatList()
    {
        # // =====
        # // | Get NBT Reference table, and collect NBT info |
        # // =====

        $This.Name      = "NbtStatList"
        $This.Reference  = $This.GetNbtStatReference()
        $This.Output     = @( )
    }
    Refresh()
    {
        $This.Output     = @( )
        $This.Count      = 0

        $This.Local()
    }
    Local()
    {
        $Stack           = nbtstat -N
        ForEach ($Line in $Stack)
        {
            Switch -Regex ($Line)
            {
                ".+\.:"
                {
                    $This.Output += $This.NbtStatInterface("Local", $Line)
                    $This.Count   = $This.Output.Count
                }
                "^Node IpAddress"
                {
                    $This.Output[-1].AddNode($Line, $Null)
                }
                Registered
                {
                    $This.Output[-1].AddHost($Line)
                    $This.Output[-1].Output[-1] | % { $_.Service = $This.Service($_) }
                }
            }
        }
    }
    Remote([Object]$Node)
    {
        $Stack           = nbtstat -A $Node.IpAddress
        ForEach ($Line in $Stack)
        {
            Switch -Regex ($Line)
            {
                ".+\.:"
                {
                    $This.Output += $This.NbtStatInterface("Remote", $Line)
                    $This.Count   = $This.Output.Count
                }
                "^Node IpAddress"
                {
                    $This.Output[-1].AddNode($Line, $Node.IpAddress)
                }
                Registered
                {
                    $This.Output[-1].AddHost($Line)
                    $This.Output[-1].Output[-1] | % { $_.Service = $This.Service($_) }
                }
            }
        }
    }
}
[String] Service([Object]$Item)

```

```

{
    Return $This.Reference | ? Id -eq $Item.Id | ? Type -eq $Item.Type | % Service
}
[Object[]] GetNbtStatReference()
{
    $Out = "00/{0}/Workstation {4};01/{0}/Messenger {6};01/{1}/Master Browser;03"+
   ("/{0}/Messenger {6};06/{0}/RAS Server {6};1F/{0}/NetDDE {6};20/{0}/File Serv"+
    "er {6};21/{0}/RAS Client {6};22/{0}/{2} Interchange(MSMail Connector);23/{0}"+
    "}/{2} Exchange Store;24/{0}/{2} Directory;30/{0}/{4} Server;31/{0}/{4} Clie"+
    "nt;43/{0}/{3} Control;44/{0}/SMS Administrators Remote Control Tool {6};45/"+
    "{0}/{3} Chat;46/{0}/{3} Transfer;4C/{0}/DEC TCPIP SVC on Windows NT;42/{0}/"+
    "mccaffee anti-virus;52/{0}/DEC TCPIP SVC on Windows NT;87/{0}/{2} MTA;6A/{0}"+
    "}/{2} IMC;BE/{0}/{5} Agent;BF/{0}/{5} Application;03/{0}/Messenger {6};00/{1}"+
    "1/{7} Name;1B/{0}/{7} Master Browser;1C/{1}/{7} Controller;1D/{0}/Master B"+
    "rowser;1E/{1}/Browser {6} Elections;2B/{0}/Lotus Notes Server;2F/{1}/Lotus "+
    "Notes ;33/{1}/Lotus Notes ;20/{1}/DCA IrmaLan Gateway Server;01/{1}/MS NetB"+
    "IOS Browse Service"

    $Out = $Out -f "UNIQUE","GROUP","Microsoft Exchange","SMS Clients Remote",
    "Modem Sharing","Network Monitor","Service","Domain"

    Return $Out -Split ";" | % { [NbtStatReference]::New($_) }
}
[Object] NbtStatInterface([String]$Type,[String]$Line)
{
    Return [NbtStatInterface]::New($Type,$Line)
}
[String] ToString()
{
    Return "({0}) <FENetwork.NbtStatList>" -f $This.Count
}
}

```

Class [NetStatAddress]

Class [NbtStatList]

```

# // =====
# // | Used for associating a netstat object |
# // =====

Class NetStatAddress
{
    [String] $IpAddress
    [String] $Port
    NetStatAddress([String]$Item)
    {
        If ($Item -match "([.+\])")
        {
            $This.IpAddress = [Regex]::Matches($Item,"([.+\])").Value
            $This.Port = $Item.Replace($This.IpAddress,"")
            $This.IpAddress = $Item.TrimStart("[").Split("%")[0]
        }

        Else
        {
            $This.IpAddress = $Item.Split(":")[0]
            $This.Port = $Item.Split(":")[1]
        }
    }
    [String] ToString()
    {
        Return "<FENetwork.NetStatAddress>"
    }
}

```

Class [NetStatAddress]

Class [NetStatConnection] /

```
# // =====
# // | Used for each line of a netstat table |
# // =====

Class NetStatConnection
{
    [String] $Protocol
    [String] $LocalAddress
    [String] $LocalPort
    [String] $RemoteAddress
    [String] $RemotePort
    [String] $State
    [String] $Direction
    NetStatConnection([String]$Line)
    {
        $Item = $Line -Split " " | ? Length -gt 0
        $This.Protocol = $Item[0]
        $This.LocalAddress = $This.GetAddress($Item[1])
        $This.LocalPort = $Item[1].Replace($This.LocalAddress + ":", "")
        $This.RemoteAddress = $This.GetAddress($Item[2])
        $This.RemotePort = $Item[2].Replace($This.RemoteAddress + ":", "")
        $This.State = $Item[3]
        $This.Direction = $Item[4]
    }
    [String] GetAddress([String]$Item)
    {
        Return @(If ($Item -match "(\\.[.\\])")
        {
            [Regex]::Matches($Item, "(\\.[.\\])").Value
        }
        Else
        {
            $Item.Split(":")[0]
        })
    }
    [String] ToString()
    {
        Return "<FENetwork.NetStatConnection>"
    }
}
```

Class [NetStatConnection]

Class [NetStatList] /

```
# // =====
# // | Parses an entire netstat table |
# // =====

Class NetStatList
{
    [Object] $Name
    [UInt32] $Count
    [Object] $Output
    NetStatList()
    {
        $This.Name = "NetStatList"
    }
    Refresh()
    {
        $This.Output = @( )
        $This.Count = 0

        $Table = netstat -ant
    }
}
```

```

        $Section      = @{}

        ForEach ($Line in $Table)
        {
            If ($Line -match "(TCP|UDP)")
            {
                $Section.Add($Section.Count,$This.NetstatConnection($Line))
            }
        }

        Switch ($Section.Count)
        {
            0
            {
            }
            1
            {
                $This.Output = $Section[0]
                $This.Count  = 1
            }
            Default
            {
                $This.Output = @($Section[0..($Section.Count-1)])
                $This.Count  = $This.Output.Count
            }
        }
    }
    [Object] NetStatConnection([String]$Line)
    {
        Return [NetStatConnection]::New($Line)
    }
    [String] ToString()
    {
        Return "{0} <FENetwork.NetStatList>" -f $This.Count
    }
}

```

```
Class [NetworkAdapterProperty]
```

```
Class [NetStatList]
```

```

# // =====
# // | Represents properties for a NetworkAdapter object |
# // =====

Class NetworkAdapterProperty
{
    [String] $Adapter
    [UInt32] $Rank
    [String] $Name
    [Object] $Value
    NetworkAdapterProperty([UInt32]$Adapter,[String]$Rank,[String]$Name,[Object]$Value)
    {
        $This.Adapter = $Adapter
        $This.Rank     = $Rank
        $This.Name     = $Name
        $This.Value    = $Value
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkAdapterProperty>"
    }
}

```

Class [NetworkAdapter]

Class [NetworkAdapterProperty]

```
# // =====
# // | Represents a NetworkAdapter object |
# // =====

Class NetworkAdapter
{
    [UInt32]    $Index
    [UInt32]    $Rank
    [String]    $Name
    [String]    $Type
    [Object]    $Property
    NetworkAdapter([Object]$Adapter)
    {
        $This.Rank      = $Adapter.DeviceId
        $This.Name      = $Adapter.Name
        $This.Type      = $Adapter.AdapterType
        $This.Property = @( )

        ForEach ($Item in $Adapter.PSObject.Properties)
        {
            $This.AddProperty($Item.Name,$Item.Value)
        }
    }
    AddProperty([String]$Name,[Object]$Value)
    {
        $This.Property += [NetworkAdapterProperty]::New($This.Rank,
                                                         $This.Property.Count,
                                                         $Name,
                                                         $Value)
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkAdapter>"
    }
}
```

Class [NetworkAdapterList]

Class [NetworkAdapter]

```
# // =====
# // | Represents a list of (0 or more) NetworkAdapter objects |
# // =====

Class NetworkAdapterList
{
    [String]    $Name
    [UInt32]    $Count
    [Object]    $Output
    NetworkAdapterList()
    {
        $This.Name      = "NetworkAdapterList"
    }
    [Object[]] Cmdlet()
    {
        Return Get-CimInstance Win32_NetworkAdapter
    }
    Refresh()
    {
        $This.Output      = @( )
        ForEach ($Adapter in $This.Cmdlet())
        {
            $This.Add($Adapter)
        }
    }
}
```

```

        $This.Output      = $This.Output | Sort-Object Rank
    }
    Add([Object]$Adapter)
    {
        $This.Output += [NetworkAdapter]::New($Adapter)
        $This.Count   = $This.Output.Count
    }
    [String] ToString()
    {
        Return "{0} <FENetwork.NetworkAdapterList>" -f $This.Count
    }
}

```

```
Class [NetworkAdapterConfigProperty] /
```

```
Class [NetworkAdapterList] /
```

```

# // =====
# // | Represents properties for a NetworkAdapterConfiguration object |
# // =====

Class NetworkAdapterConfigProperty
{
    [String] $Adapter
    [UInt32] $Rank
    [String] $Name
    [Object] $Value
    NetworkAdapterConfigProperty([UInt32]$Adapter,[String]$Rank,[String]$Name,[Object]$Value)
    {
        $This.Adapter = $Adapter
        $This.Rank     = $Rank
        $This.Name     = $Name
        $This.Value    = $Value
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkAdapterConfigProperty>"
    }
}

```

```
Class [NetworkAdapterConfig] /
```

```
Class [NetworkAdapterConfigProperty] /
```

```

# // =====
# // | Represents a NetworkAdapterConfig object |
# // =====

Class NetworkAdapterConfig
{
    [UInt32] $Index
    [UInt32] $Rank
    [String] $Name
    [String] $Service
    [UInt32] $Dhcp
    [Object] $Property
    NetworkAdapterConfig([Object]$Config)
    {
        $This.Rank     = $Config.Index
        $This.Name     = $Config.Description
        $This.Service   = $Config.ServiceName
        $This.Dhcp      = $Config.DhcpEnabled
        $This.Property = @( )
    }
}

```

```

        ForEach ($Item in $Config.PSObject.Properties)
        {
            $This.AddProperty($Item.Name,$Item.Value)
        }
    }
    AddProperty([String]$Name,[Object]$Value)
    {
        $This.Property += [NetworkAdapterConfigProperty]::New($This.Index,
                                                                $This.Property.Count,
                                                                $Name,
                                                                $Value)
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkAdapterConfig>"
    }
}

```

```

Class [NetworkAdapterConfigList]

```

```

Class [NetworkAdapterConfig]

```

```

# // =====
# // | Represents a list of (0 or more) NetworkAdapterConfig objects |
# // =====

Class NetworkAdapterConfigList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    NetworkAdapterConfigList()
    {
        $This.Name = "NetworkAdapterConfigList"
    }
    [Object[]] Cmdlet()
    {
        Return Get-CimInstance Win32_NetworkAdapterConfiguration
    }
    Refresh()
    {
        $This.Output = @( )

        ForEach ($Config in $This.Cmdlet())
        {
            $This.Add($Config)
        }
    }
    Add([Object]$Config)
    {
        $This.Output += [NetworkAdapterConfig]::New($Config)
        $This.Count = $This.Output.Count
    }
    [String] ToString()
    {
        Return "{0} <FENetwork.NetworkAdapterConfigList>" -f $This.Count
    }
}

```

```

Class [NetworkRouteProperty]

```

```

Class [NetworkAdapterConfigList]

```

```

# // =====

```



```

# // | Represents properties for a NetworkIp object |
# // =====

Class NetworkRouteProperty
{
    [UInt32] $Index
    [UInt32] $Rank
    [String] $Name
    [Object] $Value
    NetworkRouteProperty([UInt32]$Index, [UInt32]$Rank, [String]$Name, [Object]$Value)
    {
        $This.Index = $Index
        $This.Rank = $Rank
        $This.Name = $Name
        $This.Value = $Value
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkRouteProperty>"
    }
}

```

```
Class [NetworkRoute]
```

```
Class [NetworkRouteProperty]
```

```

# // =====
# // | Represents a NetworkRoute object |
# // =====

Class NetworkRoute
{
    [UInt32] $Index
    [UInt32] $Type
    [String] $DestinationPrefix
    [String] $NextHop
    [UInt32] $RouteMetric
    [String] $State
    Hidden [Object] $Property
    NetworkRoute([Object]$Route)
    {
        $This.Index = $Route.InterfaceIndex
        $This.Type = Switch -Regex ($Route.AddressFamily.ToString())
        {
            4 { 4 }
            6 { 6 }
        }
        $This.DestinationPrefix = $Route.DestinationPrefix
        $This.NextHop = $Route.NextHop
        $This.RouteMetric = $Route.RouteMetric
        $This.State = $Route.State
        $This.Property = @( )

        ForEach ($Item in $Route.PSObject.Properties)
        {
            $This.AddProperty($Item.Name, $Item.Value)
        }
    }
    AddProperty([String]$Name, [Object]$Value)
    {
        $This.Property += [NetworkRouteProperty]::New($This.Index,
                                                    $This.Property.Count,
                                                    $Name,
                                                    $Value)
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkRoute>"
    }
}

```

```
}
}
```

```
Class [NetworkRouteList] /
```

```
Class [NetworkRoute]
```

```
# // =====
# // | Represents a list of (0 or more) NetworkRoute object(s) |
# // =====

Class NetworkRouteList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    NetworkRouteList()
    {
        $This.Name = "NetworkRouteList"
    }
    [Object[]] Cmdlet()
    {
        Return Get-CimInstance MSFT_NetRoute -Namespace ROOT/StandardCimv2
    }
    Refresh()
    {
        $This.Output = @( )

        ForEach ($Route in $This.Cmdlet())
        {
            $This.Add($Route)
        }

        $This.Output = $This.Output | Sort-Object Index
    }
    Add([Object]$Item)
    {
        $This.Output += [NetworkRoute]::New($Item)
        $This.Count = $This.Output.Count
    }
    [String] ToString()
    {
        Return "{0} <FENetwork.NetworkRouteList>" -f $This.Count
    }
}
```

```
Class [NetworkInterfaceProperty] /
```

```
Class [NetworkRouteList]
```

```
# // =====
# // | Represents properties for a NetworkInterface object |
# // =====

Class NetworkInterfaceProperty
{
    [UInt32] $Index
    [UInt32] $Rank
    [UInt32] $Type
    [String] $Name
    [Object] $Value
    NetworkInterfaceProperty([UInt32]$Index, [UInt32]$Rank, [UInt32]$Type, [String]$Name, [Object]$Value)
    {
        $This.Index = $Index
        $This.Rank = $Rank
    }
}
```

```

        $This.Type = $Type
        $This.Name = $Name
        $This.Value = $Value
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkInterfaceProperty>"
    }
}

```

Class [NetworkInterface]

Class [NetworkInterfaceProperty]

```

# // =====
# // | Represents a NetworkInterface object |
# // =====

Class NetworkInterface
{
    [Object] $Index
    [String] $Alias
    [UInt32] $Type
    [UInt32] $Dhcp
    [UInt32] $Open
    [Object] $Property
    NetworkInterface([Object]$Interface)
    {
        $This.Index = $Interface.InterfaceIndex
        $This.Alias = $Interface.InterfaceAlias
        $This.Type = Switch -Regex ($Interface.AddressFamily.ToString())
        {
            4 { 4 }
            6 { 6 }
        }
        $This.Dhcp = $Interface.Dhcp -eq "Enabled"
        $This.Open = $Interface.ConnectionState -eq "Connected"
        $This.Property = @( )

        ForEach ($Item in $Interface.PSObject.Properties)
        {
            $This.AddProperty($Item.Name,$Item.Value)
        }
        AddProperty([String]$Name,[Object]$Value)
        {
            $This.Property += [NetworkInterfaceProperty]::New($This.Index,
                                                                $This.Property.Count,
                                                                $This.Type,
                                                                $Name,
                                                                $Value)
        }
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkInterface>"
    }
}

```

Class [NetworkInterfaceList]

Class [NetworkInterface]

```

# // =====
# // | Represents a list of (0 or more) NetworkInterface objects |

```

```
# // =====
Class NetworkInterfaceList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    NetworkInterfaceList()
    {
        $This.Name = "NetworkInterfaceList"
    }
    [Object[]] Cmdlet()
    {
        Return Get-CimInstance MSFT_NetIPInterface -Namespace ROOT\StandardCimv2
    }
    Refresh()
    {
        $This.Output = @( )

        ForEach ($If in $This.Cmdlet())
        {
            $This.Add($If)
        }

        $This.Output = $This.Output | Sort-Object Index
    }
    Add([Object]$If)
    {
        $This.Output += [NetworkInterface]::New($If)
        $This.Count = $This.Output.Count
    }
    [String] ToString()
    {
        Return "{0}" <FENetwork.NetworkInterfaceList> -f $This.Count
    }
}
}
```

```
Class [NetworkIpProperty]
```

```
Class [NetworkInterfaceList]
```

```
# // =====
# // | Represents properties for a NetworkIp object |
# // =====
Class NetworkIpProperty
{
    [UInt32] $Index
    [UInt32] $Rank
    [String] $Name
    [Object] $Value
    NetworkIpProperty([UInt32]$Index, [UInt32]$Rank, [String]$Name, [Object]$Value)
    {
        $This.Index = $Index
        $This.Rank = $Rank
        $This.Name = $Name
        $This.Value = $Value
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkIpProperty>"
    }
}
}
```

```
Class [NetworkIp]
```

```
Class [NetworkIpProperty]
```

```

# // =====
# // | Represents a NetworkIp object |
# // =====

Class NetworkIp
{
    [UInt32]    $Index
    [UInt32]    $Type
    [Object]    $IpAddress
    [UInt32]    $Prefix
    [Object]    $Property
    NetworkIp([Object]$Ip)
    {
        $This.Index    = $Ip.InterfaceIndex
        $This.Type      = Switch -Regex ($IP.AddressFamily.ToString())
                        {
                            4 { 4 }
                            6 { 6 }
                        }
        $This.IpAddress = $Ip.IpAddress.ToString().Split("%")[0]
        $This.Prefix    = $Ip.PrefixLength
        $This.Property  = @( )

        ForEach ($Item in $IP.PSObject.Properties)
        {
            $This.AddProperty($Item.Name,$Item.Value)
        }
    }
    AddProperty([String]$Name,[Object]$Value)
    {
        $This.Property += [NetworkIpProperty]:New($This.Index,
                                                    $This.Property.Count,
                                                    $Name,
                                                    $Value)
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkIp>"
    }
}

```

```
Class [NetworkIpList] /
```

```
Class [NetworkIp] /
```

```

# // =====
# // | Represents a list of (0 or more) NetworkIp objects |
# // =====

Class NetworkIpList
{
    [String]    $Name
    [UInt32]    $Count
    [Object]    $Output
    NetworkIpList()
    {
        $This.Name    = "NetworkIpList"
    }
    Refresh()
    {
        $This.Output = @( )

        ForEach ($Address in Get-NetIpAddress)
        {
            $This.Add($Address)
        }
    }
}

```

```

        $This.Output = $This.Output | Sort-Object Index
    }
    Add([Object]$IP)
    {
        $This.Output += [NetworkIp]::New($Ip)
        $This.Count = $This.Output.Count
    }
    [String] ToString()
    {
        Return "{0}" <FENetwork.NetworkIpList> -f $This.Count
    }
}

```

Class [V4Class] /

/ Class [NetworkIpList]

```

# // =====
# // | Class object for V4 Network(s) |
# // =====

Class V4Class
{
    [UInt32] $Index
    [String] $Label
    [String] $Name
    V4Class([UInt32]$Index,[String]$Label,[String]$Name)
    {
        $This.Index = $Index
        $This.Label = $Label
        $This.Name = $Name
    }
    [String] ToString()
    {
        Return $This.Label
    }
}

```

Class [V4ClassList] /

/ Class [V4Class]

```

# // =====
# // | List object for V4 classes |
# // =====

Class V4ClassList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    V4ClassList()
    {
        $This.Name = "V4ClassList"
        $This.Output = @( )

        ForEach ($X in 0..255)
        {
            $Item = Switch ($X)
            {
                {$_ -eq 0} { "X", "N/A" }
                {$_ -in 1..126} { "A", "Class A" }
                {$_ -eq 127} { "L", "Local" }
                {$_ -in 128..191} { "B", "Class B" }
                {$_ -in 192..223} { "C", "Class C" }
            }
        }
    }
}

```

```

        {$_ -in 224..239} { "M", "Multicast" }
        {$_ -in 240..254} { "R", "Reserved" }
        {$_ -eq 255} { "B", "Broadcast" }
    }

    $This.Add($_,$Item[0],$Item[1])
}
}
Add([UInt32]$Index,[String]$Label,[String]$Name)
{
    $This.Output += [V4Class]::New($Index,$Label,$Name)
    $This.Count = $This.Output.Count
}
[Object] Get([String]$IpAddress)
{
    Return $This.Output[$IpAddress.Split(".")[0]]
}
[String] ToString()
{
    Return "{0} <FENetwork.V4ClassList>" -f $This.Count
}
}

```

```
Class [V4PingResponse] /
```

```
Class [V4ClassList]
```

```

# // =====
# // | Object returned from a ping (sweep/scan) |
# // =====

Class V4PingResponse
{
    Hidden [UInt32] $Index
    Hidden [UInt32] $Status
    [String] $IpAddress
    [String] $Hostname
    V4PingResponse([UInt32]$Index,[String]$Address,[Object]$Reply)
    {
        $This.Index = $Index
        $This.Status = $Reply.Result.Status -match "Success"
        $This.IpAddress = $Address
    }
    GetHostname()
    {
        $This.Hostname = Try
        {
            [System.Net.Dns]::Resolve($This.IpAddress).Hostname
        }
        Catch
        {
            "<Unknown>"
        }
    }
    Domain([String]$Domain)
    {
        If ($This.Hostname -match $Domain)
        {
            $This.Hostname = ("{0}.{1}" -f $This.Hostname, $Domain)
        }
    }
    [String] ToString()
    {
        Return $This.IpAddress
    }
}

```

Class [V4Network]

Class [V4PingResponse]

```
# // =====
# // | Provisions an entire IPV4 network range, information, etc. |
# // =====

Class V4Network
{
    [String] $IpAddress
    [UInt32] $Prefix
    [String] $Class
    [String] $Netmask
    [String] $Network
    [String] $Gateway
    [String] $Range
    [String] $Broadcast
    V4Network([Object]$If)
    {
        $This.IpAddress = $If.IpAddress
        $This.Prefix     = $If.Prefix
        $This.Network    = $This.GetDestinationPrefix($If)

        If (!$This.Network)
        {
            $This.Network = "-"
        }

        $This.Gateway    = $If.Route | ? DestinationPrefix -match 0.0.0.0/0 | % NextHop

        If (!$This.Gateway)
        {
            $This.Gateway = "-"
        }

        If (!$This.Network)
        {
            $This.Network = "-"
        }
    }
    [String] GetDestinationPrefix([Object]$If)
    {
        Return $If.Route | ? DestinationPrefix -match "/$( $This.Prefix )" | % DestinationPrefix
    }
    [String] ToString()
    {
        Return "{0}/{1}" -f $This.IpAddress, $This.Prefix
    }
}
```

Class [V6Network]

Class [V4Network]

```
# // =====
# // | Provisions an entire IPV6 network* |
# // =====

Class V6Network
{
    [String] $IpAddress
    [UInt32] $Prefix
    [String] $Type
    V6Network([Object]$Interface)
    {
```



```

        $This.IpAddress = $Interface.IpAddress
        $This.Prefix    = $Interface.Prefix
        $This.Type      = Switch -Regex ($This.IpAddress)
        {
            "^fe80\:." { "Link-Local" }
            "^2001\:." { "Global" }
            Default    { "Specified" }
        }
    }
    [String] ToString()
    {
        Return ("{{0}}/{1}" -f $This.IpAddress, $This.Prefix)
    }
}

```

```
Class [NetworkControllerObjectList]
```

```
Class [V6Network]
```

```

# // =====
# // | Base class for controlling various components of each network controller template |
# // =====

Class NetworkControllerObjectList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    NetworkControllerObjectList([String]$Type)
    {
        $This.Name = $Type
        $This.Clear()
    }
    Clear()
    {
        $This.Output = @( )
        $This.Count = 0
    }
    Add([Object]$Object)
    {
        $This.Output += $Object
        $This.Count = $This.Output.Count
    }
    [String] ToString()
    {
        Return ("{{0}} {1}" -f $This.Count, ($This.Output -join ", "))
    }
}

```

```
Class [NetworkControllerInterfaceList : NetworkControllerObjectList]
```

```
Class [NetworkControllerObjectList]
```

```

# // =====
# // | For collecting total number of interfaces per (adapter/config/template) |
# // =====

Class NetworkControllerInterfaceList : NetworkControllerObjectList
{
    NetworkControllerInterfaceList([String]$Type) : base($Type)
    {
    }
}

```

```
Class [NetworkControllerInterfaceList : NetworkControllerObjectList]  
Class [NetworkControllerIpList : NetworkControllerObjectList]
```

```
# // =====  
# // | For collecting total number of IP addresses per (adapter/config/template) |  
# // =====  
  
Class NetworkControllerIpList : NetworkControllerObjectList  
{  
    NetworkControllerIpList([String]$Type) : base($Type)  
    {  
  
    }  
}
```

```
Class [NetworkControllerIpList : NetworkControllerObjectList]  
Class [NetworkControllerRouteList : NetworkControllerObjectList]
```

```
# // =====  
# // | For collecting total number of network routes per (adapter/config/template) |  
# // =====  
  
Class NetworkControllerRouteList : NetworkControllerObjectList  
{  
    NetworkControllerRouteList([String]$Type) : base($Type)  
    {  
  
    }  
}
```

```
Class [NetworkControllerRouteList : NetworkControllerObjectList]  
Class [NetworkControllerArpList : NetworkControllerObjectList]
```

```
# // =====  
# // | For collecting total number of ARP (interfaces/hosts) per (adapter/config/template) |  
# // =====  
  
Class NetworkControllerArpList : NetworkControllerObjectList  
{  
    NetworkControllerArpList([String]$Type) : base($Type)  
    {  
  
    }  
}
```

```
Class [NetworkControllerArpList : NetworkControllerObjectList]  
Class [NetworkControllerNbtStatList : NetworkControllerObjectList]
```

```
# // =====  
# // | For collecting total number of NBT (interfaces/hosts) per (adapter/config/template) |  
# // =====  
  
Class NetworkControllerNbtStatList : NetworkControllerObjectList
```

```

{
    Hidden [Object] $Reference
    NetworkControllerNbtStatList([String]$Type) : base([String]$Type)
    {
        $This.Reference = $This.GetNbtStatReference()
    }
    Refresh()
    {
        $This.Clear()
        $This.Local()
    }
    Local()
    {
        $Stack = nbtstat -N
        ForEach ($Line in $Stack)
        {
            Switch -Regex ($Line)
            {
                ".+\\: $"
                {
                    $This.Output += $This.NbtStatInterface("Local", $Line)
                    $This.Count = $This.Output.Count
                }
                "^Node IpAddress"
                {
                    $This.Output[-1].AddNode($Line, $Null)
                }
                Registered
                {
                    $This.Output[-1].AddHost($Line)
                    $This.Output[-1].Output[-1] | % { $_.Service = $This.Service($_) }
                }
            }
        }
    }
    Remote([Object]$Node)
    {
        $Stack = nbtstat -A $Node.IpAddress
        ForEach ($Line in $Stack)
        {
            Switch -Regex ($Line)
            {
                ".+\\: $"
                {
                    $This.Output += $This.NbtStatInterface("Remote", $Line)
                    $This.Count = $This.Output.Count
                }
                "^Node IpAddress"
                {
                    $This.Output[-1].AddNode($Line, $Node.IpAddress)
                }
                Registered
                {
                    $This.Output[-1].AddHost($Line)
                    $This.Output[-1].Output[-1] | % { $_.Service = $This.Service($_) }
                }
            }
        }
    }
}

[String] Service([Object]$Item)
{
    Return $This.Reference | ? ID -eq $Item.ID | ? Type -eq $Item.Type | % Service
}

[Object[]] GetNbtStatReference()
{
    $Out = "00/{0}/Workstation {4};01/{0}/Messenger {6};01/{1}/Master Browser;03"+
    "/{0}/Messenger {6};06/{0}/RAS Server {6};1F/{0}/NetDDE {6};20/{0}/File Serv"+
    "er {6};21/{0}/RAS Client {6};22/{0}/{2} Interchange(MSMail Connector);23/{0}"+
    "}/{2} Exchange Store;24/{0}/{2} Directory;30/{0}/{4} Server;31/{0}/{4} Clie"+
    "nt;43/{0}/{3} Control;44/{0}/SMS Administrators Remote Control Tool {6};45/"+
    "{0}/{3} Chat;46/{0}/{3} TransFer;4C/{0}/DEC TCPIP SVC on Windows NT;42/{0}"+
    "mccaffee anti-virus;52/{0}/DEC TCPIP SVC on Windows NT;87/{0}/{2} MTA;6A/{0}"+

```

```

    "{2} IMC;BE/{0}/{5} Agent;BF/{0}/{5} Application;03/{0}/Messenger {6};00/{1}"+
    "1/{7} Name;1B/{0}/{7} Master Browser;1C/{1}/{7} Controller;1D/{0}/Master B"+
    "rowser;1E/{1}/Browser {6} Elections;2B/{0}/Lotus Notes Server;2F/{1}/Lotus "+
    "Notes ;33/{1}/Lotus Notes ;20/{1}/DCA Irmalan Gateway Server;01/{1}/MS NetB"+
    "IOS Browse Service"

    $Out = $Out -f "UNIQUE","GROUP","Microsoft Exchange","SMS Clients Remote",
    "Modem Sharing","Network Monitor","Service","Domain"

    Return $Out -Split ";" | % { [NbtStatReference]::New($_) }
}
[Object] NbtStatInterface([String]$Type,[String]$Line)
{
    Return [NbtStatInterface]::New($Type,$Line)
}
}

```

```

Class [NetworkControllerTemplate]

```

```

Class [NetworkControllerNbtStatList : NetworkControllerObjectList]

```

```

# // =====
# // | Template object meant to assemble individual controller properties |
# // =====

Class NetworkControllerTemplate
{
    [UInt32]      $Index
    [String]      $Name
    [String]      $MacAddress
    [String]      $Vendor = "-"
    [Object]      $Adapter
    [Object]      $Config
    [Object]      $Interface
    [Object]      $IP
    [Object]      $Route
    [Object]      $Arp
    [Object]      $Nbt
    NetworkControllerTemplate([UInt32]$Index,[Object]$Adapter,[Object]$Config)
    {
        $This.Index      = $Index
        $This.Adapter     = $Adapter
        $This.Config      = $Config
        $This.Name        = $This.Get(0,"Name")
        $This.MacAddress   = $This.Get(1,"MacAddress")

        If (!$This.MacAddress)
        {
            $This.MacAddress = "-"
        }

        $This.Interface   = $This.NetworkControllerInterfaceList()
        $This.Ip           = $This.NetworkControllerIpList()
        $This.Route        = $This.NetworkControllerRouteList()
        $This.Arp          = $This.NetworkControllerArpList()
        $This.Nbt          = $This.NetworkControllerNbtStatList()
    }
    [Object] NetworkControllerInterfaceList()
    {
        Return [NetworkControllerInterfaceList]::New("Interface")
    }
    [Object] NetworkControllerIpList()
    {
        Return [NetworkControllerIpList]::New("Ip")
    }
    [Object] NetworkControllerRouteList()
    {
        Return [NetworkControllerRouteList]::New("Route")
    }
}

```

```

[Object] NetworkControllerArpList()
{
    Return [NetworkControllerArpList]::New("Arp")
}
[Object] NetworkControllerNbtStatList()
{
    Return [NetworkControllerNbtStatList]::New("Nbt")
}
[Object] Get([UInt32]$Slot,[String]$Property)
{
    If ($Slot -notin 0,1)
    {
        Throw "Invalid slot"
    }

    $Item = Switch ($Slot)
    {
        0 { $This.Adapter.Property }
        1 { $This.Config.Property }
    }

    Return $Item | ? Name -eq $Property | % Value
}
SetVendor([Object]$Vendor)
{
    $This.Vendor = $Vendor.Find($This.MacAddress)
}
[String] ToString()
{
    Return "<FENetwork.NetworkControllerTemplate>"
}
}

```

```

Class [NetworkControllerTemplateList]

```

```

Class [NetworkControllerTemplate]

```

```

# // =====
# // | List object meant to contain individual controller(s) |
# // =====

Class NetworkControllerTemplateList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    NetworkControllerTemplateList()
    {
        $This.Name = "NetworkControllerTemplateList"
    }
    Clear()
    {
        $This.Output = @( )
        $This.Count = 0
    }
    Add([Object]$Object)
    {
        $This.Output += $Object
        $This.Count = $This.Output.Count
    }
    [String] ToString()
    {
        Return "{0} <FENetwork.NetworkControllerTemplateList> -f $This.Count"
    }
}

```

```

Class [NetworkControllerTemplateList]

```

Class [NetworkControllerCompartmentProperty] /-----\

```
# // =====
# // | For fine-grained control over all various properties in each compartment |
# // =====

Class NetworkControllerCompartmentProperty
{
    [UInt32] $Index
    [String] $Source
    [String] $Name
    [Object] $Value
    NetworkControllerCompartmentProperty([UInt32]$Index, [String]$Source, [Object]$Property)
    {
        $This.Index = $Index
        $This.Source = $Source
        $This.Name = $Property.Name
        $This.Value = $Property.Value
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkControllerCompartmentProperty>"
    }
}
```

Class [NetworkControllerCompartmentV4NbtHost] /-----\

Class [NetworkControllerCompartmentProperty] /-----\

```
Class NetworkControllerCompartmentV4NbtHost
{
    [UInt32] $Index
    [IpAddress] $IpAddress
    [String] a $Name
    [String] $Id
    [String] $Type
    [String] $Service
    NetworkControllerCompartmentV4NbtHost([UInt32]$Index, [Object]$Interface, [Object]$Node)
    {
        $This.Index = $Index
        $This.IpAddress = $Interface.Node
        $This.Name = $Node.Name
        $This.Id = $Node.Id
        $This.Type = $Node.Type
        $This.Service = $Node.Service
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkControllerCompartmentV4NbtHost>"
    }
}
```

Class [NetworkControllerCompartmentV4Extension] /-----\

Class [NetworkControllerCompartmentV4NbtHost] /-----\

```
# // =====
# // | Meant to contain extensions for individual V4 Compartment |
# // =====

Class NetworkControllerCompartmentV4Extension
{
    [Object] $Arp
```

```

[Object] $Nbt
[Object] $Ping
[Object] $Host
NetworkControllerCompartmentV4Extension()
{
    $This.Arp = @( )
    $This.Nbt = @( )
    $This.Ping = @( )
    $This.Host = @( )
}
AddV4NbtHost([Object]$If,[Object]$Node)
{
    $This.Host += $This.NetworkControllerCompartmentV4NbtHost($This.Host.Count,
                                                                $If,
                                                                $Node)
}
[Object] NetworkControllerCompartmentV4NbtHost([UInt32]$Index,[Object]$If,[Object]$Node)
{
    Return [NetworkControllerCompartmentV4NbtHost]::New($Index,$If,$Node)
}
[String] ToString()
{
    Return "<FENetwork.NetworkControllerV4Extension>"
}
}

```

```

\-----/
Class [NetworkControllerCompartmentV6Extension] /-----/ Class [NetworkControllerCompartmentV4Extension]
\-----/

```

```

# // =====
# // | Meant to contain extensions for individual V6 Compartment |
# // =====

Class NetworkControllerCompartmentV6Extension
{
    [Object] $Ping
    [Object] $Host
    NetworkControllerCompartmentV6Extension()
    {
        $This.Ping = @( )
        $This.Host = @( )
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkControllerV6Extension>"
    }
}

```

```

\-----/
Class [NetworkControllerCompartmentControl] /-----/ Class [NetworkControllerCompartmentV6Extension]
\-----/

```

```

Class NetworkControllerCompartmentControl
{
    [UInt32] $Index
    [Object] $Adapter
    [Object] $Config
    [Object] $Interface
    [Object] $Ip
    [UInt32] $Type
    NetworkControllerCompartmentControl([UInt32]$Index,[Object]$Tmp,[UInt32]$Type,[Object]$If,[Object]$Ip)
    {
        $This.Index = $Index
        $This.Adapter = $Tmp.Adapter
    }
}

```

```

        $This.Config = $Tmp.Config
        $This.Type = $Type
        $This.Interface = $If
        $This.Ip = $Ip
    }
    [String] ToString()
    {
        Return "<FENetworkControllerCompartmentControl>"
    }
}

```

```
Class [NetworkControllerCompartment] /
```

```
Class [NetworkControllerCompartmentControl] /
```

```

# // =====
# // | Provides control for each individual interface + IP + AddressFamily + Route + Arp + Nbt |
# // =====

Class NetworkControllerCompartment
{
    [UInt32] $Index
    Hidden [UInt32] $Type
    Hidden [Object] $Control
    [UInt32] $InterfaceIndex
    [String] $InterfaceAlias
    [UInt32] $AddressFamily
    [UInt32] $Dhcp
    [UInt32] $Connected
    [String] $IpAddress
    [UInt32] $Prefix
    [Object] $Network
    [Object] $Route
    [Object] $Extension
    [Object] $Property
    NetworkControllerCompartment([Object]$Control)
    {
        $This.Index = $Control.Index
        $This.Control = $Control
        $This.Type = $Control.Type
        $This.InterfaceIndex = $Control.Interface.Index
        $This.InterfaceAlias = $Control.Interface.Alias
        $This.AddressFamily = $Control.Interface.Type
        $This.Dhcp = $Control.Interface.Dhcp
        $This.Connected = $Control.Interface.Open
        $This.IpAddress = $Control.IP.IpAddress
        $This.Prefix = $Control.IP.Prefix
        $This.Extension = Switch ($This.Type)
        {
            4 { $This.NetworkControllerCompartmentV4Extension() }
            6 { $This.NetworkControllerCompartmentV6Extension() }
        }

        $This.Property = @( )

        ForEach ($Item in "Adapter","Config","Interface","IP")
        {
            ForEach ($Property in $Control.$Item.Property | ? Name -notmatch Cim)
            {
                $This.AddCompartmentProperty($Item,$Property)
            }
        }
    }
    [Object] NetworkControllerCompartmentControl([Object]$Interface,[Object]$Ip)
    {
        Return [NetworkControllerCompartmentControl]::New($Interface,$Ip)
    }
    [Object] NetworkControllerCompartmentV4Extension()
    {

```



```

        Return [NetworkControllerCompartmentV4Extension]::New()
    }
    [Object] NetworkControllerCompartmentV6Extension()
    {
        Return [NetworkControllerCompartmentV6Extension]::New()
    }
    AddCompartmentProperty([String]$Source, [Object]$Property)
    {
        $This.Property += [NetworkControllerCompartmentProperty]::New($This.Property.Count,
                                                                    $Source,
                                                                    $Property)
    }
    [String] ToString()
    {
        Return "<FENetwork.NetworkControllerCompartment>"
    }
}

```

```
Class [NetworkControllerCompartmentList]
```

```
Class [NetworkControllerCompartment]
```

```

# // =====
# // | This is a list object for each individual network controller compartment |
# // =====

Class NetworkControllerCompartmentList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    NetworkControllerCompartmentList()
    {
        $This.Name = "NetworkControllerCompartmentList"
        $This.Clear()
    }
    Clear()
    {
        $This.Output = @()
        $This.Count = 0
    }
    Add([Object]$Object)
    {
        $This.Output += $Object
        $This.Count = $This.Output.Count
    }
    [String] ToString()
    {
        Return "{0} <FENetwork.NetworkControllerCompartmentList>" -f $This.Count
    }
}

```

```
Class [NetworkControllerOutputProperty]
```

```
Class [NetworkControllerCompartmentList]
```

```

# // =====
# // | Controls the information for relaying adapter information to string output |
# // =====

Class NetworkControllerOutputProperty
{
    [UInt32] $Index
    [String] $Name
    [String] $Line
}

```

```

[String[]] $Property
NetworkControllerOutputProperty([UInt32]$Index,[String]$Name,[String]$Line,[String[]]$Property)
{
    $This.Index    = $Index
    $This.Name     = $Name
    $This.Line     = $Line
    $This.Property = $Property
}
}

```

```
Class [NetworkControllerOutputList]
```

```
Class [NetworkControllerOutputProperty]
```

```

# // =====
# // | Controls the list of information to be printed to the console as string output |
# // =====

Class NetworkControllerOutputList
{
    [Object] $Output
    NetworkControllerOutputList()
    {
        $This.Output = @( )

        $This.Add("IPv4 Network Information",
            "====[ IPv4 Network Information ]=====",
            @("IpAddress","Prefix","Class","Netmask","Network","Gateway","Range","Broadcast"))

        $This.Add("IPv4 Network Route(s)",
            "====[ IPv4 Network Route(s) Table ]=====",
            @("Type","DestinationPrefix","NextHop","RouteMetric","State"))

        $This.Add("IPv4 (ARP/Address Resolution Protocol)",
            "====[ IPv4 (ARP/Address Resolution Protocol) Table ]=====",
            @("IpAddress","Physical","Type"))

        $This.Add("IPv4 (NBT/NetBEUI) Node(s)",
            "====[ IPv4 (NBT/NetBEUI) Node(s) Table ]=====",
            @("Type","Name","IpAddress","Node","Count"))

        $This.Add("IPv4 Ping Host(s)",
            "====[ IPv4 Ping Host Map Table ]=====",
            @("IpAddress","Hostname"))

        $This.Add("IPv4 (Ping + NBT/NetBEUI) Host(s) Map",
            "====[ IPv4 (Ping + NBT/NetBEUI) Host(s) Map Table ]=====",
            @("Index","IpAddress","Name","Id","Type","Service"))

        $This.Add("IPv6 Network Information",
            "====[ IPv6 Network Information ]=====",
            @("IpAddress","Prefix","Type"))

        $This.Add("IPv6 Network Route(s)",
            "====[ IPv6 Network Route(s) Table ]=====",
            @("Type","DestinationPrefix","NextHop","RouteMetric","State"))

        $This.Add("Network Statistics (UDP/TCP)",
            "====[ Network Statistics (UDP/TCP) ]=====",
            @("Protocol","LocalAddress","LocalPort","RemoteAddress","RemotePort","State","Direction"))
    }
    [Object] NetworkControllerOutputProperty([String]$Name,[String]$Line,[String[]]$Property)
    {
        Return [NetworkControllerOutputProperty]::New($This.Output.Count,
                                                    $Name,
                                                    $Line,
                                                    $Property)
    }
    Add([String]$Name,[String]$Line,[String[]]$Property)
}

```

```

    {
        $This.Output += $This.NetworkControllerOutputProperty($Name,$Line,$Property)
    }
}

```

```
Class [NetworkControllerMaster]
```

```
Class [NetworkControllerOutputList]
```

```

# // =====
# // | Network controller master allows refreshing individual items |
# // | from all of the previous classes, and provides extensions   |
# // =====

Class NetworkControllerMaster
{
    [Object]      $Mode
    [Object]      $Class
    [Object]      $Vendor
    [Object]      $Arp
    [Object]      $Nbt
    [Object]      $NetStat
    [Object]      $Adapter
    [Object]      $Config
    [Object]      $Route
    [Object]      $Interface
    [Object]      $Ip
    [Object]      $Compartment
    Hidden [Object] $Form
    NetworkControllerMaster([UInt32]$Mode)
    {
        $This.Mode      = $This.ModeList()
        $This.Mode.Select($Mode)

        $This.Class      = $This.V4ClassList()

        If ($This.Mode.Selected.Index -in 4..7)
        {
            $This.Vendor = $This.VendorList()
        }

        If ($This.Mode.Selected.Index -in 2,3,6,7)
        {
            $This.Arp      = $This.ArpList()
            $This.Nbt      = $This.NbtStatList()
        }

        If ($This.Mode.Selected.Index -in 1,3,5,7)
        {
            $This.NetStat = $This.NetStatList()
        }

        $This.Adapter      = $This.NetworkAdapterList()
        $This.Config        = $This.NetworkAdapterConfigList()
        $This.Route         = $This.NetworkRouteList()
        $This.Interface     = $This.NetworkInterfaceList()
        $This.Ip            = $This.NetworkIpList()
        $This.Compartment   = $This.NetworkControllerCompartmentList()
        $This.Form          = $This.NetworkControllerOutputList().Output
    }
    [Object] Time()
    {
        Return [Time]::New()
    }
    [Object] V4PingOptions()
    {
        Return [System.Net.NetworkInformation.PingOptions]::New()
    }
    [Object] V4PingBuffer()

```

```

{
    Return 97..119 + 97..105 | % { "0x{0:X}" -f $_ }
}
[Object] ModelList()
{
    Return [ModelList]::New()
}
[Object] V4ClassList()
{
    Return [V4ClassList]::New()
}
[Object] VendorList()
{
    Return [VendorList]::New()
}
[Object] ArpList()
{
    Return [ArpList]::New()
}
[Object] NbtStatList()
{
    Return [NbtStatList]::New()
}
[Object] NetStatList()
{
    Return [NetStatList]::New()
}
[Object] NetworkAdapterList()
{
    Return [NetworkAdapterList]::New()
}
[Object] NetworkAdapterConfigList()
{
    Return [NetworkAdapterConfigList]::New()
}
[Object] NetworkRouteList()
{
    Return [NetworkRouteList]::New()
}
[Object] NetworkInterfaceList()
{
    Return [NetworkInterfaceList]::New()
}
[Object] NetworkIpList()
{
    Return [NetworkIpList]::New()
}
[Object] NetworkControllerCompartmentList()
{
    Return [NetworkControllerCompartmentList]::New()
}
[Object] NetworkControllerTemplateList()
{
    Return [NetworkControllerTemplateList]::New()
}
[Object] NetworkControllerTemplate([UInt32]$Index,[UInt32]$Rank)
{
    Return [NetworkControllerTemplate]::New($Index,
                                            $This.Adapter.Output[$Rank],
                                            $This.Config.Output[$Rank])
}
[Object] NetworkControllerCompartmentControl([Object]$Object,[UInt32]$Type,[Object]$Tmp,[Object]$Ip)
{
    Return [NetworkControllerCompartmentControl]::New($This.Compartment.Count,$Object,$Type,$Tmp,$Ip)
}
[Object] NetworkControllerCompartment([Object]$Control)
{
    Return [NetworkControllerCompartment]::New($Control)
}
[Object] NetworkControllerOutputList()
{
    Return [NetworkControllerOutputList]::New()
}

```

```

}
[Object] V4Network([Object]$If)
{
    $Item = [V4Network]::New($If)

    $Item.Class = $This.Class.Get($Item.IPAddress)

    $Sstr = (0..31 | % { [Int32]($_ -lt $Item.Prefix); If ($_ -in 7,15,23) {"."} }) -join ''
    $Item.Netmask = ($Sstr.Split(".") | % { [Convert]::ToInt32($_,2) }) -join "."

    If (!$Item.Network)
    {
        $This.V4HostRange($Item)
        $This.V4Broadcast($Item)
    }

    Return $Item
}
[Void] V4HostRange([Object]$Item)
{
    $Item.Range = @( Switch ($Item.Network)
    {
        "-"
        {
            "N/A"
        }
        Default
        {
            $X = [UInt32[]]$Item.Network.Split("/")[0].Split(".")
            $Y = [UInt32[]]$Item.Netmask.Split(".") | % { (256 - $_) - 1 }
            @( ForEach ($I in 0..3)
            {
                Switch($Y[$I])
                {
                    0 { $X[$I] } Default { "{0}..{1}" -f $X[$I],($X[$I]+$Y[$I]) }
                }
            } ) -join '/'
        }
    })
}
[Void] V4Broadcast([Object]$Item)
{
    If ($Item.Network -ne "-")
    {
        $Split = $Item.Range.Split("/")
        $T = @{}
        0..3 | % { $T.Add($_,(Invoke-Expression $Split[$_])) }

        $Item.Broadcast = Switch -Regex ($Item.Class)
        {
            "(^A$|^Local$)" { $T[0], $T[1][-1], $T[2][-1], $T[3][-1] -join "." }
            "(^Apipa$|^MC$|^R$|^BC$)" { "-" }
            ^B$ { $T[0], $T[1], $T[2][-1], $T[3][-1] -join "." }
            ^C$ { $T[0], $T[1], $T[2], $T[3][-1] -join "." }
        }
    }
    Else
    {
        $Item.Broadcast = "-"
    }
}
[Object] V6Network([Object]$Interface)
{
    Return [V6Network]::New($Interface)
}
[Object] V4Ping([String]$Ip)
{
    $Item = [System.Net.NetworkInformation.Ping]::New()
    Return $Item.SendPingAsync($Ip,100,$This.V4PingBuffer(),$This.V4PingOptions())
}
[Object] V4PingResponse([UInt32]$Index,[Object]$Ip,[Object]$Ping)
{
}

```

```

        Return [V4PingResponse]::New($Index,$Ip,$Ping)
    }
    V4PingSweep([UInt32]$Index)
    {
        $Item = $This.Get($Index)

        If (!$Item)
        {
            Throw "Not a valid compartment index"
        }

        ElseIf ($Item.Type -ne 4)
        {
            Throw "Not a valid IPv4 compartment"
        }

        # Scratch variables/hashtables
        $Time           = $This.Time()
        $Item.Extension.Ping = @( )
        $X               = @{}
        $H               = @{}
        $P               = @{}
        $R               = @{}
        $Output          = @( )

        # Expand notation
        ForEach ($Object in $Item.Network.Range -Split "\/")
        {
            $X.Add($X.Count,($Object | Invoke-Expression))
        }

        # Populate total possible hosts from notation
        ForEach ($0 in $X[0])
        {
            ForEach ($1 in $X[1])
            {
                ForEach ($2 in $X[2])
                {
                    ForEach ($3 in $X[3])
                    {
                        $H.Add($H.Count,"$0.$1.$2.$3")
                    }
                }
            }
        }

        # Process based on the host count
        Switch ($H.Count)
        {
            0
            {
                Throw "No addresses detected"
            }
            1
            {
                # Send ping async
                Write-Host "[$Time] Scanning [~] (1) Host"
                $P.Add(0,$This.V4Ping($H[0]))

                # Await response
                Write-Host "[$Time] Sent [~] Awaiting Response"
                $R.Add(0,$This.V4PingResponse(0,$H[0],$P[0]))

                # Prepare output
                $Output = $R[0] | ? Status
            }
            Default
            {
                # Send ping async
                Write-Host "[$Time] Scanning [~] ({0}) Hosts" -f $H.Count
                ForEach ($I in 0..($H.Count-1))
                {

```

```

        $P.Add($P.Count,$This.V4Ping($H[$I]))
    }

    # Await response
    Write-Host "[${Time}] Sent [~] Awaiting Response"
    ForEach ($I in 0..($P.Count-1))
    {
        $R.Add($I,$This.V4PingResponse($I,$H[$I],$P[$I]))
    }

    # Prepare output
    $Output = $R[0..($R.Count-1)] | ? Status
}

# Show process
Write-Host "[${Time}] Scanned [+] (${Output.Count}) Host(s) repoded"

# Regardless of output count
Write-Host "[${Time}] Resolving [~] Hostnames"
ForEach ($Object in $Output)
{
    $Object.GetHostname()
}

Write-Host "[${Time}] Resolved [+] Hostnames"

# Assign the output to the IPV4 network extension property "Ping"
$Item.Extension.Ping = $Output
}
NbtScan([UInt32]$Index)
{
    $Item = $This.Get($Index)

    If (!$Item)
    {
        Throw "Not a valid compartment index"
    }

    ElseIf ($Item.Type -ne 4)
    {
        Throw "Not a valid IPv4 compartment"
    }

    # Perform ping sweep on the correct adapter
    $This.V4PingSweep($Index)

    # (Clear/reset) the Nbt table
    $Item.Extension.Nbt.Refresh()
    ForEach ($Node in $Item.Extension.Ping | ? IPAddress -notmatch $Item.IPAddress)
    {
        $Item.Extension.Nbt.Remote($Node)
    }

    # Clear host array, then build based on all nodes
    $Item.Extension.Host = @( )
    ForEach ($Node in $Item.Extension.Nbt.Output)
    {
        ForEach ($Slot in $Node.Output)
        {
            $Item.Extension.AddV4NbtHost($Node,$Slot)
        }
    }

    # Sort by IPAddress
    $Item.Extension.Host = $Item.Extension.Host | Sort-Object IPAddress

    # Rerank index
    For ($X = 0; $X -lt $Item.Extension.Host.Count; $X++)
    {
        $Item.Extension.Host[$X].Index = $X
    }
}

```

```

}
[Object[]] RefreshTemplate()
{
    # // =====
    # // | Prepare the template list |
    # // =====

    $Template = $This.NetworkControllerTemplateList()
    $Template.Clear()

    If ($This.Mode.Selected.Index -in 2,3,6,7)
    {
        $This.Arp.Refresh()
        $This.Nbt.Refresh()
        ForEach ($Item in $This.Nbt.Output)
        {
            $Item.Index = $This.Arp.Output | ? IPAddress -eq $Item.IPAddress | % Index
        }
    }

    If ($This.Mode.Selected.Index -in 1,3,5,7)
    {
        $This.NetStat.Refresh()
    }

    # // =====
    # // | Refresh all individual subcomponents (Adapter/Config/Route/Interface/Ip) |
    # // =====

    $This.Adapter.Refresh()
    $This.Config.Refresh()
    $This.Route.Refresh()
    $This.Interface.Refresh()
    $This.Ip.Refresh()

    # // =====
    # // | Filter everything into its' corresponding compartment template |
    # // =====

    $C = $This.Adapter.Output.Count
    $D = ([String]$C).Length
    $T = $C - 1

    If ($C -lt 2)
    {
        Throw "Add something to manage a 0-1 adapter(s)"
    }

    $Splat = @{
        Activity      = "Refreshing [~] Template(s)"
        Status        = "{0:d$D}/{1}" -f 0, $T
        PercentComplete = 0
    }

    Write-Progress @Splat

    ForEach ($X in 0..($This.Adapter.Output.Count-1))
    {
        $Index      = $This.Config.Output[$X].Property | ? Name -eq InterfaceIndex | % Value
        $Id          = $This.NetworkControllerTemplate($Index,$X)

        If ($This.Mode.Selected.Index -in 4..7)
        {
            If ($Id.MacAddress -ne "--")
            {
                $Id.SetVendor($This.Vendor)
            }
        }

        If ($This.Mode.Selected.Index -in 1,3,5,7)
        {

```



```

        ForEach ($Item in $This.Arp.Output | ? Index -eq $Index)
        {
            $Id.Arp.Add($Item)
        }

        ForEach ($Item in $This.Nbt.Output | ? Index -eq $Index)
        {
            $Id.Nbt.Add($Item)
        }
    }

    ForEach ($Item in $This.Interface.Output | ? Index -eq $Index)
    {
        $Id.Interface.Add($Item)
    }

    ForEach ($Item in $This.Ip.Output | ? Index -eq $Index)
    {
        $Id.IP.Add($Item)
    }

    ForEach ($Item in $This.Route.Output | ? Index -eq $Index)
    {
        $Id.Route.Add($Item)
    }

    $Template.Add($Id)

    $Splat = @{
        Activity      = "Refreshing [~] Template(s)"
        Status        = "{0:d$D}/{1}" -f $X, $T
        PercentComplete = ($X*100)/$T
    }

    Write-Progress @Splat
}

Write-Progress -Activity "Refreshing [~] Template(s)" -Complete

# // =====
# // | Sort each template by the index, then process compartment output |
# // =====

$Template.Output = $Template.Output | Sort-Object Index

Return $Template
}
Refresh()
{
    $Template = $This.RefreshTemplate()

    $This.Compartment.Clear()

    $C = $Template.Output.Count
    $D = ([String]$C).Length
    $T = $C - 1

    If ($C -lt 2)
    {
        Throw "Add something to manage a 0-1 compartments(s)"
    }

    $Splat = @{
        Activity      = "Refreshing [~] Compartment(s)"
        Status        = "{0:d$D}/{1}" -f 0, $T
        PercentComplete = 0
    }

    Write-Progress @Splat
}

```

```

ForEach ($X in 0..($Template.Output.Count-1))
{
    $Object = $Template.Output[$X]
    ForEach ($Type in 4,6)
    {
        ForEach ($Interface in $Object.Interface.Output | ? Type -eq $Type)
        {
            ForEach ($IP in $Object.IP.Output | ? Type -eq $Type)
            {
                $Control = $This.NetworkControllerCompartmentControl($Object,$Type,
$Interface,$IP)

                $Item = $This.NetworkControllerCompartment($Control)
                $Item.Route = $Object.Route.Output | ? Type -eq $Type
                If ($Type -eq 4)
                {
                    $Item.Network = $This.V4Network($Item)

                    If ($Object.Arp.Count -gt 0)
                    {
                        $Item.Extension.Arp = $Object.Arp
                    }

                    If ($Object.Nbt.Count -gt 0)
                    {
                        $Item.Extension.Nbt = $Object.Nbt
                    }
                }
                If ($Type -eq 6)
                {
                    $Item.Network = $This.V6Network($Item)
                }

                $This.Compartment.Add($Item)
            }
        }
    }

    $Splat = @{
        Activity = "Refreshing [~] Template(s)"
        Status = "({0:d}/{1})" -f $X, $T
        PercentComplete = ($X*100)/$T
    }

    Write-Progress @Splat
}

Write-Progress -Activity "Refreshing [~] Compartment(s)" -Complete
}

[Object] Section([Object]$Object,[String[]]$Names)
{
    Return New-FEFormat -Section $Object -Property $Names
}

[Object] Table([Object]$Object,[String[]]$Names)
{
    Return New-FEFormat -Table $Object -Property $Names
}

Add([Hashtable]$Hash,[Object]$Object)
{
    ForEach ($Line in $Object)
    {
        $Hash.Add($Hash.Count,$Line)
    }
}

[String[]] Draw([Hashtable]$Hashtable)
{
    Return $Hashtable[0..($Hashtable.Count-1)]
}

[String[]] List()
{
    $Out = @{}
    $Property =

```

```

"Index", "InterfaceIndex", "InterfaceAlias", "AddressFamily", "Dhcp", "Connected", "IpAddress", "Prefix"
$Section = $This.Section($This.Compartment.Output, $Property)

Switch ($This.Compartment.Count)
{
    0
    {
        Throw "No available compartments"
    }
    1
    {
        $This.Add($Out, $Section.Draw(0))
        $This.Add($Out, $This.DrawCompartment($This.Get(0)))
    }
    Default
    {
        ForEach ($X in 0..($This.Compartment.Output.Count-1))
        {
            $This.Add($Out, $Section.Draw($X))
            $This.Add($Out, $This.DrawCompartment($This.Get($X)))
        }
    }
}

Return $This.Draw($Out)
}
[Object] DrawCompartment([Object]$If)
{
    $X = $This.Form
    Return @( Switch ($If.Type)
    {
        4
        {
            # 0 / Network (IPv4)
            $X[0].Line
            $This.Table($If.Network, $X[0].Property).Draw()

            # 1 / Route
            If ($If.Route.Count -gt 0)
            {
                $X[1].Line
                $This.Table($If.Route, $X[1].Property).Draw()
            }

            # 2 / Extension.Arp
            If ($If.Extension.Arp.Count -gt 0)
            {
                $X[2].Line
                $This.Table($If.Extension.Arp.output, $X[2].Property).Draw()
            }

            # 3 / Extension.Nbt
            If ($If.Extension.Nbt.Count -gt 0)
            {
                $X[3].Line
                $This.Table($If.Extension.Nbt.Output, $X[3].Property).Draw()
            }

            # 4 / Extension.Ping
            If ($If.Extension.Ping.Count -gt 0)
            {
                $X[4].Line
                $This.Table($If.Extension.Ping, $X[4].Property).Draw()
            }

            # 5 / Extension.Host
            If ($If.Extension.Host.Count -gt 0)
            {
                $X[5].Line
                $This.Table($If.Extension.Host, $X[5].Property).Draw()
            }
        }
    })
}

```

```

        6
        {
            # 6 / Network (IPv6)
            $X[6].Line
            $This.Table($If.Network,$X[6].Property).Draw()

            # 7 / Route
            If ($If.Route.Count -gt 0)
            {
                $X[7].Line
                $This.Table($If.Route,$X[7].Property).Draw()
            }
        }
    })
}
[Object] Get([UInt32]$Index)
{
    If ($Index -gt $This.Compartment.Count)
    {
        Throw "Invalid compartment index"
    }

    Return $This.Compartment.Output[$Index]
}
}

```

```

-----/
Class [NetworkControllerMaster]

```

```

$Ctrl = [NetworkControllerMaster]::New($Mode)
$Ctrl.Refresh()
$Ctrl
}

```

```

Output /-----\
/-----

```

Since I don't have a lot of time to finish this particular document with the level of detail that I would like, TODAY...? I'm going to quickly run through what this thing does.

First and foremost,

```

PS Prompt:\> $Ctrl = Get-FENetwork -Mode 7
PS Prompt:\> $Ctrl

Mode          : 7
Class         : (256) <FENetwork.V4ClassList>
Vendor        : (28664) <FENetwork.VendorList>
Arp           : (2) <FENetwork.ArpList>
Nbt           : (5) <FENetwork.NbtStatList>
NetStat       : (92) <FENetwork.NetStatList>
Adapter       : (21) <FENetwork.NetworkAdapterList>
Config        : (21) <FENetwork.NetworkAdapterConfigList>
Route         : (39) <FENetwork.NetworkRouteList>
Interface     : (12) <FENetwork.NetworkInterfaceList>
Ip            : (12) <FENetwork.NetworkIpList>
Compartment   : (10) <FENetwork.NetworkControllerCompartmentList>

```

```

PS Prompt:\>

```

```

PS Prompt:\> $Ctrl.Mode

```

```

Name      Count Output      Selected
-----
Modelist   8 {0, 1, 2, 3...} 7

```

```

PS Prompt:\>

```

```

PS Prompt:\> $Ctrl.Mode.Output

```

```
Index Type Description
-----
0 None [ ] Vendor [ ] Arp/Nbt [ ] Netstat
1 NetstatOnly [ ] Vendor [ ] Arp/Nbt [X] Netstat
2 ArpNbtOnly [ ] Vendor [X] Arp/Nbt [ ] Netstat
3 ArpNbtNetstat [ ] Vendor [X] Arp/Nbt [X] Netstat
4 VendorOnly [X] Vendor [ ] Arp/Nbt [ ] Netstat
5 VendorNetstat [X] Vendor [ ] Arp/Nbt [X] Netstat
6 VendorArpNbt [X] Vendor [X] Arp/Nbt [ ] Netstat
7 All [X] Vendor [X] Arp/Nbt [X] Netstat

PS Prompt:\> $Ctrl.Class.Output

Index Label Name
-----
0 X N/A
1 A Class A
2 A Class A
3 A Class A
...
239 M Multicast
240 R Reserved
241 R Reserved
242 R Reserved
...
253 R Reserved
254 R Reserved
255 B Broadcast

PS Prompt:\>

PS Prompt:\> $Ctrl.Vendor

Name Count Output
-----
VendorList 28664 {<FENetwork.VendorItem>, <FENetwork.VendorItem>, <FENetwork.VendorItem>...}

PS Prompt:\>

PS Prompt:\> $Ctrl.Arp

Name Count Output
-----
ArpList 2 {<FENetwork.ArpAdapter>, <FENetwork.ArpAdapter>}

PS Prompt:\>

PS Prompt:\> $Ctrl.Arp.Output

Index Type IPAddress Host
-----
23 Public 10.1.99.144 {<FENetwork.ArpHost>, <FENetwork.ArpHost>, <FENetwork.ArpHost>...}
2 Public 172.28.128.1 {<FENetwork.ArpHost>, <FENetwork.ArpHost>, <FENetwork.ArpHost>...}

PS Prompt:\>

PS Prompt:\> $Ctrl.Arp.Output[0].Host

IPAddress Physical Type
-----
10.1.99.1 18-c2-41-02-5a-4c Host
10.1.99.255 ff-ff-ff-ff-ff-ff HostMax
224.0.0.2 01-00-5e-00-00-02 Multicast
224.0.0.22 01-00-5e-00-00-16 Multicast
224.0.0.251 01-00-5e-00-00-fb Multicast
224.0.0.252 01-00-5e-00-00-fc Multicast
239.255.255.250 01-00-5e-7f-ff-fa Multicast
255.255.255.255 ff-ff-ff-ff-ff-ff Broadcast

PS Prompt:\>

PS Prompt:\> $Ctrl.Nbt.Output | FT

Index Type Name IPAddress Node Count Output
```

```

-----
0 Local Ethernet 0.0.0.0 0.0.0.0 0 {}
2 Local vEthernet (Default Switch) 172.28.128.1 172.28.128.1 3 {<FENetwork.NbtStatHost>...}
23 Local Wi-Fi 10.1.99.144 10.1.99.144 3 {<FENetwork.NbtStatHost>...}
0 Local Local Area Connection* 1 0.0.0.0 0.0.0.0 0 {}
0 Local Local Area Connection* 2 0.0.0.0 0.0.0.0 0 {}

PS Prompt:\>

PS Prompt:\> $Ctrl.Nbt.Output[1].Output | FT

Index Name Id Type Service
-----
0 L420-X64 <20> UNIQUE File Server Service
1 L420-X64 <00> UNIQUE Workstation Modem Sharing
2 SECURED <00> GROUP Domain Name

PS Prompt:\>

PS Prompt:\> $Ctrl.Netstat

Name Count Output
-----
NetStatList 92 {<FENetwork.NetStatConnection>, <FENetwork.NetStatConnection>...}

PS Prompt:\>

PS Prompt:\> $Ctrl.Netstat.Output | FT

Protocol LocalAddress LocalPort RemoteAddress RemotePort State Direction
-----
TCP 0.0.0.0 135 0.0.0.0 0 LISTENING InHost
TCP 0.0.0.0 445 0.0.0.0 0 LISTENING InHost
TCP 0.0.0.0 2179 0.0.0.0 0 LISTENING InHost
TCP 0.0.0.0 3389 0.0.0.0 0 LISTENING InHost
TCP 0.0.0.0 5040 0.0.0.0 0 LISTENING InHost
TCP 0.0.0.0 5985 0.0.0.0 0 LISTENING InHost
...
UDP [::] 61867 * *
UDP [::1] 1900 * *
UDP [::1] 5353 * *
UDP [::1] 56672 * *
UDP [fe80::91c7:fc1c:f8eb:d470%23] 1900 * *
UDP [fe80::91c7:fc1c:f8eb:d470%23] 56671 * *
UDP [fe80::b9a3:f67d:1d89:4b%42] 1900 * *
UDP [fe80::b9a3:f67d:1d89:4b%42] 56670 * *

PS Prompt:\>

PS Prompt:\> $Ctrl.Adapter

Name Count Output
-----
NetworkAdapterList 21 {<FENetwork.NetworkAdapter>, <FENetwork.NetworkAdapter>...}

PS Prompt:\>

PS Prompt:\> $Ctrl.Adapter.Output[0]

Index : 0
Rank : 0
Name : Microsoft Kernel Debug Network Adapter
Type :
Property : {<FENetwork.NetworkAdapterProperty>, <FENetwork.NetworkAdapterProperty>...}

PS Prompt:\>

PS Prompt:\> $Ctrl.Adapter.Output[0].Property

Adapter Rank Name Value
-----
0 0 Caption [00000000] Microsoft Kernel Debug Network Adapter
0 1 Description Microsoft Kernel Debug Network Adapter
0 2 InstallDate
0 3 Name Microsoft Kernel Debug Network Adapter

```

```

0      4 Status
0      5 Availability 3
0      6 ConfigManagerErrorCode 0
0      7 ConfigManagerUserConfig False
0      8 CreationClassName Win32_NetworkAdapter
0      9 DeviceID 0
0     10 ErrorCleared
0     11 ErrorDescription
0     12 LastErrorCode
0     13 PNPDeviceID ROOT\KDNIC\0000
0     14 PowerManagementCapabilities
0     15 PowerManagementSupported False
0     16 StatusInfo
0     17 SystemCreationClassName Win32_ComputerSystem
0     18 SystemName L420-X64
0     19 AutoSense
0     20 MaxSpeed
0     21 NetworkAddresses
0     22 PermanentAddress
0     23 Speed
0     24 AdapterType
0     25 AdapterTypeId
0     26 GUID
0     27 Index 0
0     28 Installed True
0     29 InterfaceIndex 6
0     30 MACAddress
0     31 Manufacturer Microsoft
0     32 MaxNumberControlled 0
0     33 NetConnectionID
0     34 NetConnectionStatus
0     35 NetEnabled
0     36 PhysicalAdapter False
0     37 ProductName Microsoft Kernel Debug Network Adapter
0     38 ServiceName kdnic
0     39 TimeOfLastReset 12/21/2022 10:08:11 AM
0     40 PSComputerName
0     41 CimClass root/cimv2:Win32_NetworkAdapter
0     42 CimInstanceProperties {Caption, Description, InstallDate, Name...}
0     43 CimSystemProperties Microsoft.Management.Infrastructure.CimSystemProperties

```

PS Prompt:\>

PS Prompt:\> **\$Ctrl.Config**

```

Name          Count Output
----
NetworkAdapterConfigList 21 {<FENetwork.NetworkAdapterConfig>, <FENetwork.NetworkAdapterConfig>...}

```

PS Prompt:\>

PS Prompt:\> **\$Ctrl.Config.Output | Format-Table**

Index	Rank	Name	Service	Dhcp	Property
0	0	Microsoft Kernel Debug Network Adapter	kdnic	1	{<FENetwork.Network...
0	1	1x1 11bgn Wireless LAN PCI Express Half Mini Card Adapter	rtwlane_13	1	{<FENetwork.Network...
0	2	Realtek PCIe GbE Family Controller	rt640x64	1	{<FENetwork.Network...
0	3	Intel(R) Ethernet Connection I217-LM	eli65x64	1	{<FENetwork.Network...
0	4	Microsoft Wi-Fi Direct Virtual Adapter	vwifimp	1	{<FENetwork.Network...
0	5	Microsoft Wi-Fi Direct Virtual Adapter	vwifimp	0	{<FENetwork.Network...
0	6	WAN Miniport (SSTP)	RasSstp	0	{<FENetwork.Network...
0	7	WAN Miniport (IKEv2)	RasAgileVpn	0	{<FENetwork.Network...
0	8	WAN Miniport (L2TP)	RasL2tp	0	{<FENetwork.Network...
0	9	WAN Miniport (PPTP)	PptpMiniport	0	{<FENetwork.Network...
0	10	WAN Miniport (PPPOE)	RasPppoe	0	{<FENetwork.Network...
0	11	WAN Miniport (IP)	NdisWan	0	{<FENetwork.Network...
0	12	WAN Miniport (IPv6)	NdisWan	0	{<FENetwork.Network...
0	13	WAN Miniport (Network Monitor)	NdisWan	0	{<FENetwork.Network...
0	14	RAS Async Adapter	AsyncMac	0	{<FENetwork.Network...
0	15	Intel(R) Centrino(R) Advanced-N 6235 Driver	NETwNe64	1	{<FENetwork.Network...
0	16	Microsoft Wi-Fi Direct Virtual Adapter	vwifimp	1	{<FENetwork.Network...
0	17	Remote NDIS based Internet Sharing Device	usbrndis6	1	{<FENetwork.Network...

0	18	Hyper-V Virtual Switch Extension Adapter	VMSMP	0 {<FENetwork.Network...
0	19	Hyper-V Virtual Ethernet Adapter	VMSNPXYMP	0 {<FENetwork.Network...
0	20	Remote NDIS based Internet Sharing Device	usbrndis6	1 {<FENetwork.Network...

PS Prompt:\>

PS Prompt:\> \$Ctrl.Config.Output[0].Property

Adapter	Rank	Name	Value
-----	----	----	----
0	0	Caption	[00000000] Microsoft Kernel Debug Network Adapter
0	1	Description	Microsoft Kernel Debug Network Adapter
0	2	SettingID	{27021307-ADC3-4967-B1BB-09A0A87E4F50}
0	3	ArpAlwaysSourceRoute	
0	4	ArpUseEtherSNAP	
0	5	DatabasePath	
0	6	DeadGWDetectEnabled	
0	7	DefaultIPGateway	
0	8	DefaultTOS	
0	9	DefaultTTL	
0	10	DHCPEnabled	True
0	11	DHCPLeaseExpires	
0	12	DHCPLeaseObtained	
0	13	DHCPServer	
0	14	DNSDomain	
0	15	DNSDomainSuffixSearchOrder	
0	16	DNSEnabledForWINSResolution	
0	17	DNSHostName	
0	18	DNSServerSearchOrder	
0	19	DomainDNSRegistrationEnabled	
0	20	ForwardBufferMemory	
0	21	FullDNSRegistrationEnabled	
0	22	GatewayCostMetric	
0	23	IGMPLevel	
0	24	Index	0
0	25	InterfaceIndex	6
0	26	IPAddress	
0	27	IPConnectionMetric	
0	28	IPEnabled	False
0	29	IPFilterSecurityEnabled	
0	30	IPPortSecurityEnabled	
0	31	IPSecPermitIPProtocols	
0	32	IPSecPermitTCPPorts	
0	33	IPSecPermitUDPPorts	
0	34	IPSubnet	
0	35	IPUseZeroBroadcast	
0	36	IPXAddress	
0	37	IPXEnabled	
0	38	IPXFrameType	
0	39	IPXMediaType	
0	40	IPXNetworkNumber	
0	41	IPXVirtualNetNumber	
0	42	KeepAliveInterval	
0	43	KeepAliveTime	
0	44	MACAddress	
0	45	MTU	
0	46	NumForwardPackets	
0	47	PMTUBHDetectEnabled	
0	48	PMTUDiscoveryEnabled	
0	49	ServiceName	kdnic
0	50	TcpipNetbiosOptions	
0	51	TcpMaxConnectRetransmissions	
0	52	TcpMaxDataRetransmissions	
0	53	TcpNumConnections	
0	54	TcpUserRFC1122UrgentPointer	
0	55	TcpWindowSize	
0	56	WINSEnableLMHostsLookup	
0	57	WINSHostLookupFile	
0	58	WINSPrimaryServer	
0	59	WINSScopeID	
0	60	WINSSecondaryServer	
0	61	PSComputerName	
0	62	CimClass	root/cimv2:Win32_NetworkAdapterConfiguration


```
0      63 CimInstanceProperties      {Caption, Description, SettingID, ArpAlwaysSourceRoute...}
0      64 CimSystemProperties        Microsoft.Management.Infrastructure.CimSystemProperties
```

PS Prompt:\>

PS Prompt:\> **\$Ctrl.Route**

```
Name          Count Output
----          -
NetworkRouteList 39 {<FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...}
```

PS Prompt:\>

PS Prompt:\> **\$Ctrl.Route.Output | Format-Table**

Index	Type	DestinationPrefix	NextHop	RouteMetric	State
1	4	127.255.255.255/32	0.0.0.0	256	Alive
1	4	255.255.255.255/32	0.0.0.0	256	Alive
1	4	127.0.0.1/32	0.0.0.0	256	Alive
1	4	127.0.0.0/8	0.0.0.0	256	Alive
1	6	:::1/128	::	256	Alive
1	4	224.0.0.0/4	0.0.0.0	256	Alive
1	6	ff00::/8	::	256	Alive
14	4	224.0.0.0/4	0.0.0.0	256	Alive
14	6	fe80::/64	::	256	Alive
14	6	ff00::/8	::	256	Alive
14	6	fe80::9fdf:e88:c051:8d9d/128	::	256	Alive
14	4	255.255.255.255/32	0.0.0.0	256	Alive
15	6	fe80::25ea:80b7:ee6e:a606/128	::	256	Alive
15	4	224.0.0.0/4	0.0.0.0	256	Alive
15	4	255.255.255.255/32	0.0.0.0	256	Alive
15	6	fe80::/64	::	256	Alive
15	6	ff00::/8	::	256	Alive
22	6	ff00::/8	::	256	Alive
22	4	255.255.255.255/32	0.0.0.0	256	Alive
22	6	fe80::e417:59fd:4600:db3/128	::	256	Alive
22	4	224.0.0.0/4	0.0.0.0	256	Alive
22	6	fe80::/64	::	256	Alive
23	6	fe80::/64	::	256	Alive
23	6	fe80::91c7:fc1c:f8eb:d470/128	::	256	Alive
23	6	ff00::/8	::	256	Alive
23	4	10.1.99.144/32	0.0.0.0	256	Alive
23	4	10.1.99.255/32	0.0.0.0	256	Alive
23	4	224.0.0.0/4	0.0.0.0	256	Alive
23	4	0.0.0.0/0	10.1.99.1	0	Alive
23	4	255.255.255.255/32	0.0.0.0	256	Alive
23	4	10.1.99.0/24	0.0.0.0	256	Alive
42	4	172.28.143.255/32	0.0.0.0	256	Alive
42	4	255.255.255.255/32	0.0.0.0	256	Alive
42	4	224.0.0.0/4	0.0.0.0	256	Alive
42	6	fe80::b9a3:f67d:1d89:4b/128	::	256	Alive
42	6	ff00::/8	::	256	Alive
42	4	172.28.128.0/20	0.0.0.0	256	Alive
42	6	fe80::/64	::	256	Alive
42	4	172.28.128.1/32	0.0.0.0	256	Alive

PS Prompt:\>

PS Prompt:\> **\$Ctrl.Interface**

```
Name          Count Output
----          -
NetworkInterfaceList 12 {<FENetwork.NetworkInterface>, <FENetwork.NetworkInterface>...}
```

PS Prompt:\>

PS Prompt:\> **\$Ctrl.Interface.Output | Format-Table**

Index	Alias	Type	Dhcp	Open	Property	
1	Loopback	Pseudo-Interface	1	4	0	1 {<FENetwork.NetworkInterfaceProperty>...}
1	Loopback	Pseudo-Interface	1	6	0	1 {<FENetwork.NetworkInterfaceProperty>...}
14	Ethernet		4	1	0	{<FENetwork.NetworkInterfaceProperty>...}

```

14 Ethernet 6 1 0 {<FENetwork.NetworkInterfaceProperty>...}
15 Local Area Connection* 1 4 1 0 {<FENetwork.NetworkInterfaceProperty>...}
15 Local Area Connection* 1 6 0 0 {<FENetwork.NetworkInterfaceProperty>...}
22 Local Area Connection* 2 4 0 0 {<FENetwork.NetworkInterfaceProperty>...}
22 Local Area Connection* 2 6 1 0 {<FENetwork.NetworkInterfaceProperty>...}
23 Wi-Fi 6 1 1 {<FENetwork.NetworkInterfaceProperty>...}
23 Wi-Fi 4 1 1 {<FENetwork.NetworkInterfaceProperty>...}
42 vEthernet (Default Switch) 6 1 1 {<FENetwork.NetworkInterfaceProperty>...}
42 vEthernet (Default Switch) 4 0 1 {<FENetwork.NetworkInterfaceProperty>...}

```

PS Prompt:\>

PS Prompt:\> \$Ctrl.Interface.Output[0].Property | Format-Table

Index	Rank	Type	Name	Value
-----	----	----	----	-----
1	0	4	Store	ActiveStore
1	1	4	AddressFamily	IPv4
1	2	4	Forwarding	Disabled
1	3	4	ClampMss	Disabled
1	4	4	Advertising	Disabled
1	5	4	NeighborUnreachabilityDetection	Disabled
1	6	4	RouterDiscovery	ControlledByDHCP
1	7	4	NeighborDiscoverySupported	No
1	8	4	ManagedAddressConfiguration	Enabled
1	9	4	OtherStatefulConfiguration	Enabled
1	10	4	WeakHostSend	Disabled
1	11	4	WeakHostReceive	Disabled
1	12	4	IgnoreDefaultRoutes	Disabled
1	13	4	AdvertiseDefaultRoute	Disabled
1	14	4	ForceArpNdWolPattern	Disabled
1	15	4	DirectedMacWolPattern	Disabled
1	16	4	EcnMarking	AppDecide
1	17	4	Dhcp	Disabled
1	18	4	ConnectionState	Connected
1	19	4	AutomaticMetric	Enabled
1	20	4	ifIndex	1
1	21	4	ifAlias	Loopback Pseudo-Interface 1
1	22	4	Caption	
1	23	4	Description	
1	24	4	ElementName	
1	25	4	InstanceID	
1	26	4	CommunicationStatus	
1	27	4	DetailedStatus	
1	28	4	HealthState	
1	29	4	InstallDate	
1	30	4	Name	;55<55;
1	31	4	OperatingStatus	
1	32	4	OperationalStatus	
1	33	4	PrimaryStatus	
1	34	4	Status	
1	35	4	StatusDescriptions	
1	36	4	AvailableRequestedStates	
1	37	4	EnabledDefault	2
1	38	4	EnabledState	
1	39	4	OtherEnabledState	
1	40	4	RequestedState	12
1	41	4	TimeOfLastStateChange	
1	42	4	TransitioningToState	12
1	43	4	CreationClassName	
1	44	4	SystemCreationClassName	
1	45	4	SystemName	
1	46	4	NameFormat	
1	47	4	OtherTypeDescription	
1	48	4	ProtocolIFType	
1	49	4	ProtocolType	
1	50	4	AliasAddresses	
1	51	4	GroupAddresses	
1	52	4	LANID	
1	53	4	LANType	
1	54	4	MACAddress	
1	55	4	MaxDataSize	
1	56	4	OtherLANType	

```

1 57 4 AdvertisedRouterLifetime 00:30:00
1 58 4 BaseReachableTime 30000
1 59 4 CompartmentId 1
1 60 4 CurrentHopLimit 0
1 61 4 DadRetransmitTime 1000
1 62 4 DadTransmits 0
1 63 4 InterfaceAlias Loopback Pseudo-Interface 1
1 64 4 InterfaceIndex 1
1 65 4 InterfaceMetric 75
1 66 4 IsolationId 0
1 67 4 LowestIfNetLuid 0
1 68 4 NLMTu 4294967295
1 69 4 ReachableTime 44000
1 70 4 RetransmitTime 1000
1 71 4 PSComputerName
1 72 4 CimClass ROOT/StandardCimv2:MSFT_NetIPInterface
1 73 4 CimInstanceProperties {Caption, Description, ElementName, InstanceID...}
1 74 4 CimSystemProperties Microsoft.Management.Infrastructure.CimSystemProperties

PS Prompt:\>

PS Prompt:\> $Ctrl.Ip

Name Count Output
----
NetworkIpList 12 {<FENetwork.NetworkIp>, <FENetwork.NetworkIp>, <FENetwork.NetworkIp>...}

PS Prompt:\>

PS Prompt:\> $Ctrl.Ip.Output | FT

Index Type IpAddress Prefix Property
-----
1 4 127.0.0.1 8 {<FENetwork.NetworkIpProperty>...}
1 6 ::1 128 {<FENetwork.NetworkIpProperty>...}
14 4 169.254.144.82 16 {<FENetwork.NetworkIpProperty>...}
14 6 fe80::9fdf:e88:c051:8d9d 64 {<FENetwork.NetworkIpProperty>...}
15 4 169.254.232.98 16 {<FENetwork.NetworkIpProperty>...}
15 6 fe80::25ea:80b7:ee6e:a606 64 {<FENetwork.NetworkIpProperty>...}
22 4 169.254.190.64 16 {<FENetwork.NetworkIpProperty>...}
22 6 fe80::e417:59fd:4600:db3 64 {<FENetwork.NetworkIpProperty>...}
23 6 fe80::91c7:fc1c:f8eb:d470 64 {<FENetwork.NetworkIpProperty>...}
23 4 10.1.99.144 24 {<FENetwork.NetworkIpProperty>...}
42 6 fe80::b9a3:f67d:1d89:4b 64 {<FENetwork.NetworkIpProperty>...}
42 4 172.28.128.1 20 {<FENetwork.NetworkIpProperty>...}

PS Prompt:\>

PS Prompt:\> $Template = $Ctrl.RefreshTemplate()
PS Prompt:\> $Template

Name Count Output
----
NetworkControllerTemplateList 21 {<FENetwork.NetworkControllerTemplate>...}

PS Prompt:\>

PS Prompt:\> $Template.Output

Index : 2
Name : WAN Miniport (IP)
MacAddress : 30:FF:20:52:41:53
Vendor : -
Adapter : <FENetwork.NetworkAdapter>
Config : <FENetwork.NetworkAdapterConfig>
Interface : (0)
IP : (0)
Route : (0)
Arp : (1) <FENetwork.ArpAdapter>
Nbt : (1) <FENetwork.NbtStatInterface>

Index : 4
Name : WAN Miniport (Network Monitor)

```

```
MacAddress : 5C:E1:20:52:41:53
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 6
Name        : Microsoft Kernel Debug Network Adapter
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 8
Name        : Remote NDIS based Internet Sharing Device
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 9
Name        : WAN Miniport (SSTP)
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 10
Name        : WAN Miniport (PPPOE)
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 11
Name        : Intel(R) Ethernet Connection I217-LM
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 12
Name        : RAS Async Adapter
```

```
MacAddress : 20:41:53:59:4E:FF
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 13
Name        : WAN Miniport (L2TP)
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 14
Name        : Realtek PCIe GbE Family Controller
MacAddress  : 04:7D:7B:5F:D5:45
Vendor      : Quanta
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (2) <FENetwork.NetworkInterface>, <FENetwork.NetworkInterface>
IP          : (2) <FENetwork.NetworkIp>, <FENetwork.NetworkIp>
Route       : (5) <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...
Arp         : (0)
Nbt         : (0)

Index       : 15
Name        : Microsoft Wi-Fi Direct Virtual Adapter
MacAddress  : 9C:B7:0D:20:08:FE
Vendor      : Liteon
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (2) <FENetwork.NetworkInterface>, <FENetwork.NetworkInterface>
IP          : (2) <FENetwork.NetworkIp>, <FENetwork.NetworkIp>
Route       : (5) <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...
Arp         : (0)
Nbt         : (0)

Index       : 16
Name        : WAN Miniport (IPv6)
MacAddress  : 40:4D:20:52:41:53
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 17
Name        : WAN Miniport (PPTP)
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 19
Name        : Remote NDIS based Internet Sharing Device
```

```
MacAddress : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 20
Name        : WAN Miniport (IKEv2)
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 21
Name        : Hyper-V Virtual Switch Extension Adapter
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 22
Name        : Microsoft Wi-Fi Direct Virtual Adapter #2
MacAddress  : 9C:B7:0D:20:08:FE
Vendor      : Liteon
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (2) <FENetwork.NetworkInterface>, <FENetwork.NetworkInterface>
IP          : (2) <FENetwork.NetworkIp>, <FENetwork.NetworkIp>
Route       : (5) <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...
Arp         : (0)
Nbt         : (0)

Index       : 23
Name        : 1x1 11bgn Wireless LAN PCI Express Half Mini Card Adapter
MacAddress  : 9C:B7:0D:20:08:FE
Vendor      : Liteon
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (2) <FENetwork.NetworkInterface>, <FENetwork.NetworkInterface>
IP          : (2) <FENetwork.NetworkIp>, <FENetwork.NetworkIp>
Route       : (9) <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...
Arp         : (1) <FENetwork.ArpAdapter>
Nbt         : (1) <FENetwork.NbtStatInterface>

Index       : 24
Name        : Microsoft Wi-Fi Direct Virtual Adapter
MacAddress  : -
Vendor      : -
Adapter     : <FENetwork.NetworkAdapter>
Config      : <FENetwork.NetworkAdapterConfig>
Interface   : (0)
IP          : (0)
Route       : (0)
Arp         : (0)
Nbt         : (0)

Index       : 25
Name        : Intel(R) Centrino(R) Advanced-N 6235 Driver
```

```
MacAddress : -
Vendor     : -
Adapter    : <FENetwork.NetworkAdapter>
Config     : <FENetwork.NetworkAdapterConfig>
Interface  : (0)
IP         : (0)
Route      : (0)
Arp        : (0)
Nbt        : (0)
```

```
Index      : 42
Name       : Hyper-V Virtual Ethernet Adapter
MacAddress : 00:15:5D:AB:5A:4B
Vendor     : Microsoft
Adapter    : <FENetwork.NetworkAdapter>
Config     : <FENetwork.NetworkAdapterConfig>
Interface  : (2) <FENetwork.NetworkInterface>, <FENetwork.NetworkInterface>
IP         : (2) <FENetwork.NetworkIp>, <FENetwork.NetworkIp>
Route      : (8) <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...
Arp        : (0)
Nbt        : (0)
```

PS Prompt:\>

PS Prompt:\> **\$Template**.Output[10]

```
Index      : 15
Name       : Microsoft Wi-Fi Direct Virtual Adapter
MacAddress : 9C:B7:0D:20:08:FE
Vendor     : Liteon
Adapter    : <FENetwork.NetworkAdapter>
Config     : <FENetwork.NetworkAdapterConfig>
Interface  : (2) <FENetwork.NetworkInterface>, <FENetwork.NetworkInterface>
IP         : (2) <FENetwork.NetworkIp>, <FENetwork.NetworkIp>
Route      : (5) <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...
Arp        : (0)
Nbt        : (0)
```

PS Prompt:\> **\$Template**.Output[10].Adapter

```
Index      : 0
Rank       : 4
Name       : Microsoft Wi-Fi Direct Virtual Adapter
Type       : Ethernet 802.3
Property   : {<FENetwork.NetworkAdapterProperty>, <FENetwork.NetworkAdapterProperty>...}
```

PS Prompt:\> **\$Template**.Output[10].Config

```
Index      : 0
Rank       : 4
Name       : Microsoft Wi-Fi Direct Virtual Adapter
Service    : vwifimp
Dhcp       : 1
Property   : {<FENetwork.NetworkAdapterConfigProperty>, <FENetwork.NetworkAdapterConfigProperty>...}
```

PS Prompt:\> **\$Template**.Output[10].Interface

Name	Count	Output
Interface	2	{<FENetwork.NetworkInterface>, <FENetwork.NetworkInterface>}

PS Prompt:\> **\$Template**.Output[10].Ip

Name	Count	Output
Ip	2	{<FENetwork.NetworkIp>, <FENetwork.NetworkIp>}

PS Prompt:\> **\$Template**.Output[10].Ip.Route

PS Prompt:\> **\$Template**.Output[10].Route

Name	Count	Output
Route	5	{<FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...}

```
PS Prompt:\>
```

The templates are not actually kept, because most of these adapters have no underlying connections or activity. There's no need to keep all of this information in memory, UNLESS, it is needed, then it can be collected via `$Template = $Ctrl.RefreshTemplate()`

```
PS Prompt:\> $Ctrl.Compartment
```

```
Name                               Count Output
----                               -
NetworkControllerCompartmentList  10 {<FENetwork.NetworkControllerCompartment>...}
```

```
PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Compartment.Output | Format-Table (Edited)
```

Index	InterfaceIndex	InterfaceAlias	AddressFamily	Dhcp	Connected	IpAddress	Prefix
0	14	Ethernet	4	1	0	169.254.144.82	16
1	14	Ethernet	6	1	0	fe80::9fdf:e88:c051:8d9d	64
2	15	Local Area Connection* 1	4	1	0	169.254.232.98	16
3	15	Local Area Connection* 1	6	0	0	fe80::25ea:80b7:ee6e:a606	64
4	22	Local Area Connection* 2	4	0	0	169.254.190.64	16
5	22	Local Area Connection* 2	6	1	0	fe80::e417:59fd:4600:dbe3	64
6	23	Wi-Fi	4	1	1	10.1.99.144	24
7	23	Wi-Fi	6	1	1	fe80::91c7:fc1c:f8eb:d470	64
8	42	vEthernet (Default Switch)	4	0	1	172.28.128.1	20
9	42	vEthernet (Default Switch)	6	1	1	fe80::b9a3:f67d:1d89:4b	64

```
PS Prompt:\> $Ctrl.Get(6)
```

```
Index          : 6
InterfaceIndex : 23
InterfaceAlias : Wi-Fi
AddressFamily  : 4
Dhcp           : 1
Connected      : 1
IpAddress      : 10.1.99.144
Prefix         : 24
Network        : 10.1.99.144/24
Route          : {<FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>, <FENetwork.NetworkRoute>...}
Extension      : <FENetwork.NetworkControllerV4Extension>
Property       : {<FENetwork.NetworkControllerCompartmentProperty>...}
```

```
PS Prompt:\>
```

Since this adapter is an Ipv4 address, then it will have the Ipv4 extension class instead of the Ipv6 extension.

The point of all of this, is to provide functionality once all of the above information is loaded into memory. Running all of these commands is rather tedious. That doesn't mean that they're not useful. It's just that having to run through all of these commands and then sort them, filter them, and then getting a useful snapshot of the necessary information you'd need, to do some REAL (system/network) administration...

All of this stuff is incredibly useful.

However, what is the point of this utility, specifically...?

Well, Active Directory domain controller promotion requires having a really good idea of what's on the network. If a system has multiple adapters and multiple interfaces, multiple IP addresses, then it can be pretty complicated to establish a connection to an ADDS domain to log in and promote a domain controller.

NBTSTAT has limited functionality in that you have to know what remote IP addresses to scan for, which you could get from the ARP table, but these tools have to be used by a HUMAN in order to understand what the correct fields are to populate and allow login.

I mean, unless of course you're trying to automate all of this stuff, and create a utility that's rather rock solid like Wireshark is. Some people might not know it, but the default utility that allows DCPromo to look for other domain controllers to authenticate...? It is actually pretty complex. Perhaps not quite as complex as all of this above information, but still...


```
PS Prompt:\> $Ctrl.Get(6).Extension
```

```
Arp          Nbt          Ping Host
---          ---          ----
(1) <FENetwork.ArpAdapter> (1) <FENetwork.NbtStatInterface> {} {}
```

```
PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.NbtScan(6)
[00:00:00.0490105] Scanning [~] (256) Hosts
[00:00:00.6140005] Sent [~] Awaiting Response
[00:00:00.8520046] Scanned [+] (2) Host(s) reponded
[00:00:00.8550078] Resolving [~] Hostnames
[00:00:05.6277143] Resolved [+] Hostnames
PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Get(6).Extension.Nbt.Output | FT
```

Index	Type	Name	IpAddress	Node	Count	Output
0	Local	Ethernet	0.0.0.0	0.0.0.0	0	{}
0	Local	vEthernet (Default Switch)	172.28.128.1	172.28.128.1	3	{<FENetwork.NbtStatHost>...}
0	Local	Wi-Fi	10.1.99.144	10.1.99.144	3	{<FENetwork.NbtStatHost>...}
0	Local	Local Area Connection* 1	0.0.0.0	0.0.0.0	0	{}
0	Local	Local Area Connection* 2	0.0.0.0	0.0.0.0	0	{}
0	Remote	Ethernet	0.0.0.0	10.1.99.1	0	{}
0	Remote	vEthernet (Default Switch)	172.28.128.1	10.1.99.1	0	{}
0	Remote	Wi-Fi	10.1.99.144	10.1.99.1	0	{}
0	Remote	Local Area Connection* 1	0.0.0.0	10.1.99.1	0	{}
0	Remote	Local Area Connection* 2	0.0.0.0	10.1.99.1	0	{}

```
PS Prompt:\>
```

Some of this information is rather redundant, and that's specifically what I've been working on, to prevent duplication of these fields and stuff. However, it is rather obvious that it is being quite thorough.

```
PS Prompt:\> $Ctrl.Get(6).Extension | Format-List
```

```
Arp : (1) <FENetwork.ArpAdapter>
Nbt : (10) <FENetwork.NbtStatInterface>, <FENetwork.NbtStatInterface>, <FENetwork.NbtStatInterface>...
Ping : {10.1.99.1, 10.1.99.144}
Host : {<FENetwork.NetworkControllerCompartmentV4NbtHost>, <FENetwork.NetworkControllerCompartmentV4NbtHost>...}
```

```
PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Get(6).Extension.Ping
```

IpAddress	Hostname
10.1.99.1	10.1.99.1
10.1.99.144	l420-x64.securedigitsplus.com

```
PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Get(6).Extension.Host | FT
```

Index	IpAddress	Name	Id	Type	Service
0	10.1.99.144	L420-X64	<00>	UNIQUE	Workstation Modem Sharing
1	10.1.99.144	L420-X64	<20>	UNIQUE	File Server Service
2	10.1.99.144	SECURED	<00>	GROUP	Domain Name
3	172.28.128.1	L420-X64	<20>	UNIQUE	File Server Service
4	172.28.128.1	L420-X64	<00>	UNIQUE	Workstation Modem Sharing
5	172.28.128.1	SECURED	<00>	GROUP	Domain Name

```
PS Prompt:\>
```

And this right here, is the information necessary to scan for available domain controllers. The specific <Id> that says <1B> or <1C> means that those (2) things are...

```
PS Prompt:\> $Ctrl.Nbt.Reference
```

ID	Type	Service
----	------	---------

```
-- ----
<00> UNIQUE Workstation Modem Sharing
<01> UNIQUE Messenger Service
<01> GROUP Master Browser
<03> UNIQUE Messenger Service
<06> UNIQUE RAS Server Service
<1F> UNIQUE NetDDE Service
<20> UNIQUE File Server Service
<21> UNIQUE RAS Client Service
<22> UNIQUE Microsoft Exchange Interchange(MSMail Connector)
<23> UNIQUE Microsoft Exchange Exchange Store
<24> UNIQUE Microsoft Exchange Directory
<30> UNIQUE Modem Sharing Server
<31> UNIQUE Modem Sharing Client
<43> UNIQUE SMS Clients Remote Control
<44> UNIQUE SMS Administrators Remote Control Tool Service
<45> UNIQUE SMS Clients Remote Chat
<46> UNIQUE SMS Clients Remote Transfer
<4C> UNIQUE DEC TCPIP SVC on Windows NT
<42> UNIQUE mccafee anti-virus
<52> UNIQUE DEC TCPIP SVC on Windows NT
<87> UNIQUE Microsoft Exchange MTA
<6A> UNIQUE Microsoft Exchange IMC
<BE> UNIQUE Network Monitor Agent
<BF> UNIQUE Network Monitor Application
<03> UNIQUE Messenger Service
<00> GROUP Domain Name
<1B> UNIQUE Domain Master Browser
<1C> GROUP Domain Controller
<1D> UNIQUE Master Browser
<1E> GROUP Browser Service Elections
<2B> UNIQUE Lotus Notes Server
<2F> GROUP Lotus Notes
<33> GROUP Lotus Notes
<20> GROUP DCA IrmaLan Gateway Server
<01> GROUP MS NetBIOS Browse Service

PS Prompt:\>
```

Either a <Domain Master Browser>, or a <Domain Controller>.

Now, what if I want to get a really quick glance at all of the information above, similar to <ipconfig>...?

```
PS Prompt:\> $Ctrl.List()

-----
| Index: 0 | InterfaceIndex: 14 | InterfaceAlias: Ethernet | AddressFamily: 4 | Dhcp: 1 | Connected: 0 | IPAddress: 169.254.144.82 | Prefix: 16 |
-----
===[ IPv4 Network Information ]=====
|-----|
| IPAddress | Prefix | Class | Netmask | Network | Gateway | Range | Broadcast |
|-----|
| 169.254.144.82 | 16 | B | 255.255.0.0 | - | - | N/A | - |
|-----|
===[ IPv4 Network Route(s) Table ]=====
|-----|
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|
| 4 | 224.0.0.0/4 | 0.0.0.0 | 256 | Alive |
| 4 | 255.255.255.255/32 | 0.0.0.0 | 256 | Alive |
|-----|

-----
| Index: 1 | InterfaceIndex: 14 | InterfaceAlias: Ethernet | AddressFamily: 6 | Dhcp: 1 | Connected: 0 | IPAddress: fe80::9fdf:e88:c851:8d9d | Prefix: 64 |
-----
===[ IPv6 Network Information ]=====
|-----|
| IPAddress | Prefix | Type |
|-----|
| fe80::9fdf:e88:c851:8d9d | 64 | Link-Local |
|-----|
===[ IPv6 Network Route(s) Table ]=====
|-----|
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|
| 6 | fe80::/64 | :: | 256 | Alive |
| 6 | ff00::/8 | :: | 256 | Alive |
| 6 | fe80::9fdf:e88:c851:8d9d/128 | :: | 256 | Alive |
|-----|

-----
| Index: 2 | InterfaceIndex: 15 | InterfaceAlias: Local Area Connection* 1 | AddressFamily: 4 | Dhcp: 1 | Connected: 0 | IPAddress: 169.254.232.98 | Prefix: 16 |
-----
===[ IPv4 Network Information ]=====
|-----|
| IPAddress | Prefix | Class | Netmask | Network | Gateway | Range | Broadcast |
|-----|
| 169.254.232.98 | 16 | B | 255.255.0.0 | - | - | N/A | - |
|-----|
===[ IPv4 Network Route(s) Table ]=====
|-----|
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|
| 4 | 224.0.0.0/4 | 0.0.0.0 | 256 | Alive |
| 4 | 255.255.255.255/32 | 0.0.0.0 | 256 | Alive |
|-----|
```

```
| Index: 3 | InterfaceIndex: 15 | InterfaceAlias: Local Area Connection* 1 | AddressFamily: 6 | Dhcp: 0 | Connected: 0 | IpAddress: fe80::25ea:80b7:ee6e:a606 | Prefix: 64 |
=====
===[ IPv6 Network Information ]=====
| IpAddress | Prefix | Type |
|-----|-----|-----|
| fe80::25ea:80b7:ee6e:a606 | 64 | Link-Local |
=====
===[ IPv6 Network Route(s) Table ]=====
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|-----|-----|-----|-----|
| 6 | fe80::25ea:80b7:ee6e:a606/128 | :: | 256 | Alive |
| 6 | fe80::/64 | :: | 256 | Alive |
| 6 | ff00::/8 | :: | 256 | Alive |
=====
| Index: 4 | InterfaceIndex: 22 | InterfaceAlias: Local Area Connection* 2 | AddressFamily: 4 | Dhcp: 0 | Connected: 0 | IpAddress: 169.254.190.64 | Prefix: 16 |
=====
===[ IPv4 Network Information ]=====
| IpAddress | Prefix | Class | Netmask | Network | Gateway | Range | Broadcast |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 169.254.190.64 | 16 | B | 255.255.0.0 | - | - | N/A | - |
=====
===[ IPv4 Network Route(s) Table ]=====
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|-----|-----|-----|-----|
| 4 | 255.255.255.255/32 | 0.0.0.0 | 256 | Alive |
| 4 | 224.0.0.0/4 | 0.0.0.0 | 256 | Alive |
=====
| Index: 5 | InterfaceIndex: 22 | InterfaceAlias: Local Area Connection* 2 | AddressFamily: 6 | Dhcp: 1 | Connected: 0 | IpAddress: fe80::e417:59fd:4600:dbe3 | Prefix: 64 |
=====
===[ IPv6 Network Information ]=====
| IpAddress | Prefix | Type |
|-----|-----|-----|
| fe80::e417:59fd:4600:dbe3 | 64 | Link-Local |
=====
===[ IPv6 Network Route(s) Table ]=====
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|-----|-----|-----|-----|
| 6 | ff00::/8 | :: | 256 | Alive |
| 6 | fe80::e417:59fd:4600:dbe3/128 | :: | 256 | Alive |
| 6 | fe80::/64 | :: | 256 | Alive |
=====
| Index: 6 | InterfaceIndex: 23 | InterfaceAlias: Wi-Fi | AddressFamily: 4 | Dhcp: 1 | Connected: 1 | IpAddress: 10.1.99.144 | Prefix: 24 |
=====
===[ IPv4 Network Information ]=====
| IpAddress | Prefix | Class | Netmask | Network | Gateway | Range | Broadcast |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 10.1.99.144 | 24 | A | 255.255.255.0 | 10.1.99.0/24 | 10.1.99.1 | 10/1/99/0..255 | 10.1.99.255 |
=====
===[ IPv4 Network Route(s) Table ]=====
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|-----|-----|-----|-----|
| 4 | 10.1.99.144/32 | 0.0.0.0 | 256 | Alive |
| 4 | 10.1.99.255/32 | 0.0.0.0 | 256 | Alive |
| 4 | 224.0.0.0/4 | 0.0.0.0 | 256 | Alive |
| 4 | 0.0.0.0/0 | 10.1.99.1 | 0 | Alive |
| 4 | 255.255.255.255/32 | 0.0.0.0 | 256 | Alive |
| 4 | 10.1.99.0/24 | 0.0.0.0 | 256 | Alive |
=====
===[ IPv4 (ARP/Address Resolution Protocol) Table ]=====
| IpAddress | Physical | Type |
|-----|-----|-----|
| 10.1.99.144 | Public |
=====
===[ IPv4 (NBT/NetBEUI) Node(s) Table ]=====
| Type | Name | IpAddress | Node | Count |
|-----|-----|-----|-----|-----|
| Local | Ethernet | 0.0.0.0 | 0.0.0.0 | 10 |
| Local | vEthernet (Default Switch) | 172.28.128.1 | 172.28.128.1 |
| Local | Wi-Fi | 10.1.99.144 | 10.1.99.144 |
| Local | Local Area Connection* 1 | 0.0.0.0 | 0.0.0.0 |
| Local | Local Area Connection* 2 | 0.0.0.0 | 0.0.0.0 |
| Remote | Ethernet | 0.0.0.0 | 10.1.99.1 |
| Remote | vEthernet (Default Switch) | 172.28.128.1 | 10.1.99.1 |
| Remote | Wi-Fi | 10.1.99.144 | 10.1.99.1 |
| Remote | Local Area Connection* 1 | 0.0.0.0 | 10.1.99.1 |
| Remote | Local Area Connection* 2 | 0.0.0.0 | 10.1.99.1 |
=====
===[ IPv4 Ping Host Map Table ]=====
| IpAddress | Hostname |
|-----|-----|
| 10.1.99.1 | 10.1.99.1 |
| 10.1.99.144 | 1420-x64.securedigitsplus.com |
=====
===[ IPv4 (Ping + NBT/NetBEUI) Host(s) Map Table ]=====
| Index | IpAddress | Name | Id | Type | Service |
|-----|-----|-----|-----|-----|-----|
| 0 | 10.1.99.144 | L420-X64 | <00> | UNIQUE | Workstation Modem Sharing |
| 1 | 10.1.99.144 | L420-X64 | <20> | UNIQUE | File Server Service |
| 2 | 10.1.99.144 | SECURED | <00> | GROUP | Domain Name |
| 3 | 172.28.128.1 | L420-X64 | <20> | UNIQUE | File Server Service |
| 4 | 172.28.128.1 | L420-X64 | <00> | UNIQUE | Workstation Modem Sharing |
| 5 | 172.28.128.1 | SECURED | <00> | GROUP | Domain Name |
=====
| Index: 7 | InterfaceIndex: 23 | InterfaceAlias: Wi-Fi | AddressFamily: 6 | Dhcp: 1 | Connected: 1 | IpAddress: fe80::91c7:fc1c:f8eb:d470 | Prefix: 64 |
=====
===[ IPv6 Network Information ]=====
| IpAddress | Prefix | Type |
|-----|-----|-----|
| fe80::91c7:fc1c:f8eb:d470 | 64 | Link-Local |
=====
===[ IPv6 Network Route(s) Table ]=====
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|-----|-----|-----|-----|
| 6 | fe80::/64 | :: | 256 | Alive |
| 6 | fe80::91c7:fc1c:f8eb:d470/128 | :: | 256 | Alive |
| 6 | ff00::/8 | :: | 256 | Alive |
=====
```

```

| Index: 8 | InterfaceIndex: 42 | InterfaceAlias: vEthernet (Default Switch) | AddressFamily: 4 | Dhcp: 0 | Connected: 1 | IpAddress: 172.28.128.1 | Prefix: 28 |
=====
===[ IPv4 Network Information ]=====
| IpAddress | Prefix | Class | Netmask | Network | Gateway | Range | Broadcast |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 172.28.128.1 | 28 | B | 255.255.248.0 | 172.28.128.0/28 | - | 172/28/128.143/0..255 | 172.28.143.255 |
=====
===[ IPv4 Network Route(s) Table ]=====
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|-----|-----|-----|-----|
| 4 | 172.28.143.255/32 | 0.0.0.0 | 256 | Alive |
| 4 | 255.255.255.255/32 | 0.0.0.0 | 256 | Alive |
| 4 | 224.0.0.0/4 | 0.0.0.0 | 256 | Alive |
| 4 | 172.28.128.0/28 | 0.0.0.0 | 256 | Alive |
| 4 | 172.28.128.1/32 | 0.0.0.0 | 256 | Alive |
=====
| Index: 9 | InterfaceIndex: 42 | InterfaceAlias: vEthernet (Default Switch) | AddressFamily: 6 | Dhcp: 1 | Connected: 1 | IpAddress: fe80::b9a3:f67d:1d89:4b | Prefix: 64 |
=====
===[ IPv6 Network Information ]=====
| IpAddress | Prefix | Type |
|-----|-----|-----|
| fe80::b9a3:f67d:1d89:4b | 64 | Link-Local |
=====
===[ IPv6 Network Route(s) Table ]=====
| Type | DestinationPrefix | NextHop | RouteMetric | State |
|-----|-----|-----|-----|-----|
| 6 | fe80::b9a3:f67d:1d89:4b/128 | :: | 256 | Alive |
| 6 | ff00::/8 | :: | 256 | Alive |
| 6 | fe80::/64 | :: | 256 | Alive |
=====
PS Prompt:\>

```

Normally, I would edit this information so it is more visible.
But since I'm strapped on time, people may have to (pinch/zoom) or what have you to see the details.

This is a rather extensive snapshot of the information for this particular system, and being able to cleanly write all of this stuff to the console is a challenge in and of itself. Being able to get this utility to work FAST, and be very thorough and not bog down the systems it is used on, is another challenge entirely.

That's what I've been having to do with each iteration of the utility and the module.

-----/

/ Conclusion /-----/ Output

This utility is a critical piece to the rest of the module, and there were many design implementations that I had to (make/amend/change) over previous iterations, in order for it to be resilient, and scalable. At some point I will clear up any potential minor issues that remain, but- those issues are impacting the function from working.

Simply put, this needs to work on EVERY SINGLE MACHINE, and, every single adapter, interface, IP address, configuration, route, etc.

The next objective is completing the Get-FEDCPromo utility, as I have already been updating THAT, to accommodate the changes made to [\[FightingEntropy\(π\)\]](#). I've also been updating the function New-FEInfrastructure, and that tool utilizes these functions. So, simply put, this function is absolutely critical to be done correctly.

Last but not least, this utility can use different modes, as I've been updating the functions for (verbosity/logging) levels.

-----/

/ Conclusion

Michael C. Cook Sr.
Security Engineer
Secure Digits Plus LLC

