```
 ____    __________________________________________________________________________________________________
//¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
\\__//¯¯¯ Get-FESystem [~] (12/30/2022)                                                              ___//¯¯\\
 ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
     ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
```

```
\______________________________________________________________________________________________________________________/
  Introduction /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Greetings,

In this document, I'm going to talk about a function that I have been developing that is able to be used in a similar manner to [msinfo32.exe], though to be clear, this function is not quite as complex as that.

The point of this function, is to be able to capture system information (regardless of the system), and then export that information to a file that can be imported on another machine.

This is effectively the precursor to other things I intend to develop for the module, [FightingEntropy(π)].

Some of this also touches on various aspects related to (DISM/Deployment and Imaging Service Module).
Some of this also touches on event logging.
Some of this also touches on things that [Laplink PCMover] does.

There will be many additional features in the future, and they'll be added to this function for various modes for either SIMPLE and FAST information, or COMPLEX and THOROUGH information.

The goal is to be able to specifically create the instructions that can be pulled from the system, and report the necessary stuff needed to identify a unique machine, operating system, hardware, installed updates, installed Windows optional features, applications, events, tasks, and any AppX based programs.

But also, any information that would be helpful for the processor, hard drives, and network adapters.

Lastly, [Laplink PCMover], which is a tool that is very similar to the (USMT/User State Migration Tool), which both of these tools are meant to assist in extracting and facilitating a transfer of a users environment.

(UEV/User Environment Virtualization) is a concept that these both touch on, where a users environment is considered equivalent whether the environment is being virtualized or not.

Perhaps I'm not describing it correctly, but the way I have always seen these utilities, is to perform seamless migration tasks, in order to transfer a users (environment/profile/settings) to a target machine, whether that is through upgrading Windows, installing updates, or converting a machine from (P2V/physical-to-virtual).

Some of these things are done via the [Microsoft Deployment Toolkit] by the geniuses over at [Microsoft].

Below, I will put the function as it is currently written, without the classes, so it can be seen that it is wrapping around the classes, and THEY do most of the heavily lifting and work.

Then, I will go over each of the classes within the function.

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

 //==================================================================================================\\
//  Module     : [FightingEntropy()][2022.12.0]                                                       \\
\\  Date       : 2022-12-30 10:01:45                                                                  //
 \\==================================================================================================//

    FileName   : Get-FESystem.ps1
    Solution   : [FightingEntropy()][2022.12.0]
    Purpose    : This function performs multiple operations in order to either collect/export,
                 or import system information from a file, or memory object
                 [+] system snapshot
                 [+] BIOS information
                 [+] computer system
                 [+] operating system
                 [+] hotfixes
                 [+] features
                 [+] applications
                 [+] events
                 [+] tasks
                 [+] processor(s)
                 [+] disk(s)
                 [+] network(s)
    Author     : Michael C. Cook Sr.
    Contact    : @mcc85s
    Primary    : @mcc85s
    Created    : 2022-12-14
    Modified   : 2022-12-30
    Demo       : N/A
    Version    : 0.0.0 - () - Finalized functional version 1
    TODO       : Switch to more (flexible/sophisticated) section platform

.Example
#>

Function Get-FESystem
{
    [CmdLetBinding(DefaultParameterSetName=0)]
    Param(
        [ValidateSet(0,1,2)]
        [Parameter(Mandatory,ParameterSetName=0)]
        [Parameter(Mandatory,ParameterSetName=1)]
        [Parameter(Mandatory,ParameterSetName=2)][UInt32]$Mode,
        [ValidateScript({Test-Path $_})]
        [Parameter(Mandatory,ParameterSetName=1)][String]$Path,
        [Parameter(Mandatory,ParameterSetName=2)][Object]$InputObject
    )

    # // ========================
    # // | <Insert classes here> |
    # // ========================

    If (!$Path -and !$InputObject)
    {
        [System]::New()
    }

    Else
    {
        If ($Path)
        {
            $InputObject = [System.IO.File]::ReadAllLines($Path)
        }

        $Section = [InputObject]::New($InputObject)
        [System]::New($Section)
    }
}
```

```
                                                                                                          ______________/
\________________________________________________________________________________________________________/ Introduction
  Class [SystemProperty] /----------------------------------------------------------------------------------------------\
/----------------------
```

```

    # // ============================================
    # // | For scalable control over the output file |
    # // ============================================

    Class SystemProperty
    {
        [UInt32]  $Index
        [UInt32]   $Rank
        [String] $Source
        [String]   $Name
        [UInt32] $Buffer
        [Object]  $Value
        SystemProperty([UInt32]$Index,[UInt32]$Rank,[String]$Source,[String]$Name,[Object]$Value)
        {
            $This.Index  = $Index
            $This.Rank   = $Rank
            $This.Source = $Source
            $This.Name   = $Name
            $This.Buffer = $Name.Length
            $This.Value  = $Value
        }
        SetBuffer([UInt32]$Width)
        {
            If ($This.Buffer -lt $Width)
            {
                $This.Buffer = $Width
            }
        }
        [String] ToString()
        {
            Return "{0} {1}" -f $This.Name.PadRight($This.Buffer," "), $This.Value
        }
    }
```

```
                                                                                                  ______________________/
\________________________________________________________________________________________________/ Class [SystemProperty]
  Class [InputLine] /---------------------------------------------------------------------------------------------------\
/-----------------
```

```
    # // ==========================================
    # // | Classifies each line of an input file |
    # // ==========================================

    Class InputLine
    {
        [Int32]         $Rank = -1
        [UInt32]       $Index
        [String]        $Slot
        [String]        $Line
        InputLine([UInt32]$Index,[String]$Line)
        {
            $This.Index = $Index
            $This.Slot  = $This.GetSlot($Line)
            $This.Line  = $Line.TrimEnd(" ")
        }
        [String] GetSlot($Line)
        {
            Return @( Switch -Regex ($Line)
            {
                "^ $"                             { "Space"    }
                "^\={120}$"                       { "Line"     }
                "^\[\w+\]$"                       { "Header"   }
                "^[a-zA-Z]+\d+\s+$"               { "Label"    }
                "^[a-zA-Z]+\d+.[a-zA-Z]+\d+\s*$" { "Sublabel" }
                "^[a-zA-Z]+\s+\:"                 { "Property" }
                "^[a-zA-Z]+\d+\s+\:"              { "Item"     }
            })
        }
        [String] ToString()
        {
            Return $This.Line
        }
    }
```

```
                                                                                                      _________________/
\_____________________________________________________________________________________________________/ Class [InputLine]
  Class [InputProperty] /-----------------------------------------------------------------------------------------------\
/-----------------------
```

```
    # // =====================================================================
    # // | Organizes properties for individual input file (sections + items) |
    # // =====================================================================

    Class InputProperty
    {
        [UInt32]  $Rank
        [String]  $Name
        [String] $Value
        InputProperty([UInt32]$Rank,[Object]$Property)
        {
            $This.Rank  = $Rank
            $This.Name  = $Property.Name
            $This.Value = $Property.Value
        }
    }
```

```
                                                                                                   _____________________/
\_________________________________________________________________________________________________/ Class [InputProperty]
  Class [InputItem] /---------------------------------------------------------------------------------------------------\
/-------------------
```

```
    # // =======================================================
    # // | Contains nested items for sections of an input file |
    # // =======================================================

    Class InputItem
    {
        [UInt32]     $Rank
        [String]     $Name
        [Object] $Property
        InputItem([UInt32]$Rank,[String]$Name)
        {
            $This.Rank     = $Rank
            $This.Name     = $Name
            $This.Property = @( )
        }
    }
```

```
                                                                                                        _________________/
\______________________________________________________________________________________________________/ Class [InputItem]
  Class [InputSection] /------------------------------------------------------------------------------------------------\
/----------------------
```

```
    # // ==============================================
    # // | Contains a single section of an input file |
    # // ==============================================

    Class InputSection
    {
        [UInt32]       $Index
        [String]        $Name
        [UInt32]        $Slot
        [Object]    $Property
        [Object]        $Item
        InputSection([UInt32]$Index,[Object[]]$Content)
        {
            $This.Index    = $Index
            $This.Name     = ($Content | ? Slot -eq Header).Line -Replace "(\[|\])",""
            $This.Property = @( )
            $This.Item     = @( )

            $Hash          = @{

                Label      = $Content | ? Slot -match Label
                Item       = $Content | ? Slot -match Item
            }
            # If the content is (1) object with multiple properties
            If ($Hash.Label.Count -eq 0 -and $Hash.Item.Count -eq 0)
            {
                $This.Slot = 0

                ForEach ($Item in $Content | ? Slot -eq Property)
                {
                    $Note           = $This.NoteProperty($Item.Line)
                    $This.Property += $This.InputProperty($This.Property.Count,$Note)
                }
            }

            # If the content is (1+) objects with (1) property
            If ($Hash.Item.Count -gt 0)
            {
                $This.Slot = 1

                ForEach ($Item in $Content | ? Slot -eq Item)
                {
                    $Note           = $This.NoteProperty($Item.Line)
                    $This.Item     += $This.InputProperty($This.Item.Count,$Note)
                }
            }

            # If the content is numerous items with their own properties
            If ($Hash.Label.Count -gt 0)
            {
                $This.Slot               = 2

                ForEach ($Item in $Content)
                {
                    If ($Item.Slot -match "Label")
                    {
                        $This.Item      += $This.InputItem($This.Item.Count,$Item.Line)
                    }

                    If ($Item.Slot -match "Property")
                    {
                        $Note            = $This.NoteProperty($Item.Line)
                        $xItem           = $This.LastItem()
                        $xItem.Property += $This.InputProperty($xItem.Property.Count,$Note)
                    }
                }
            }
        }
        [Object] LastItem()
        {
            If ($This.Item.Count -eq 0)
            {
                Throw "No current items"
            }

            Return $This.Item[$This.Item.Count-1]
        }
        [Object] NoteProperty([String]$Line)
        {
            $X = $Line.IndexOf(":")[0]
            $Y = $Line.Substring(0,$X).TrimEnd(" ")
            $Z = $Line.Substring($X+1).TrimStart(" ")

            Return [PSNoteProperty]::New($Y,$Z)
        }
        [String] GetItemName([String]$Line)
        {
            Return [Regex]::Matches($Line,"[a-zA-Z]+[0-9]+").Value
        }
        [String] GetName([String]$Line)
        {
            Return [Regex]::Matches($Line,"^\w+").Value
        }
        [String] GetValue([String]$Line)
        {
            Return $Line -Replace "^\w+\s+:\s*",""
        }
        [Object] InputProperty([UInt32]$Rank,[Object]$Property)
        {
            Return [InputProperty]::New($Rank,$Property)
        }
        [Object] InputItem([UInt32]$Rank,[String]$Name)
        {
            Return [InputItem]::New($Rank,$Name)
        }
    }
```

```
                                                                                                     ____________________/
\___________________________________________________________________________________________________/ Class [InputSection]
  Class [InputObject] /--------------------------------------------------------------------------------------------------\
/---------------------
```

```
# // =====================================
# // | Turns an input file into an object |
# // =====================================

Class InputObject
{
    [Object] $Process
    [Object] $Section
    InputObject([Object]$Content)
    {
        $This.Process = @( )
        $This.Section = @( )

        $Rank = -1
        ForEach ($Line in $Content)
        {
            $Item = $This.InputLine($This.Process.Count,$Line)

            If ($Item.Slot -eq "Header")
            {
                $Rank ++
                $This.Process[-1].Rank = $Rank
            }

            $Item.Rank = $Rank

            $This.Add($Item)

            $This.Separate()
        }
    }
    [Object] InputLine([UInt32]$Index,[String]$Line)
    {
        Return [InputLine]::New($Index,$Line)
    }
    [Object] InputSection([UInt32]$Index,[Object[]]$Content)
    {
        Return [InputSection]::New($Index,$Content)
    }
    Add([Object]$Object)
    {
        $This.Process += $Object
    }
    Separate()
    {
        $Range = $This.Process | Select-Object Rank -Unique | % Rank
        ForEach ($X in $Range)
        {
            $Slot          = $This.Process | ? Rank -eq $X
            $This.Section += $This.InputSection($X,$Slot)
        }
    }
    [Object] Get([UInt32]$Index)
```

```
        {
            $Name = Switch ($Index)
            {
                0  { "Snapshot"         }
                1  { "BiosInformation" }
                2  { "ComputerSystem"  }
                3  { "OperatingSystem" }
                4  { "HotFix"          }
                5  { "Feature"         }
                6  { "Application"     }
                7  { "Event"           }
                8  { "Task"            }
                9  { "AppX"            }
                10 { "Processor"       }
                11 { "Disk"            }
                12 { "Network"         }
            }

            Return $This.Section | ? Name -eq $Name
        }
    }
```

```
                                                                                                     ___________________/
\___________________________________________________________________________________________________/ Class [InputObject]
  Class [OutputProperty] /----------------------------------------------------------------------------------------------\
/------------------------
```

```

    # // ================================
    # // | For preparing an output file |
    # // ================================

    Class OutputProperty
    {
        [String] $Source
        [UInt32] $Rank
        [Object] $Name
        [Object] $Value
        OutputProperty([String]$Source,[UInt32]$Rank,[String]$Name,[Object]$Value)
        {
            $This.Source  = $Source
            $This.Rank    = $Rank
            $This.Name    = $Name
            $This.Value   = $Value
        }
        [String] ToString()
        {
            Return "<FESystem.OutputProperty>"
        }
    }

```

```
                                                                                                  ______________________/
\________________________________________________________________________________________________/ Class [OutputProperty]
  Class [OutputSection] /-----------------------------------------------------------------------------------------------\
/-----------------------
```

```

    # // ========================================================================
    # // | Allows each section of an output file to be formatted with precision |
    # // ========================================================================

    Class OutputSection
    {
        [UInt32]  $Index
        [String]   $Name
        [UInt32]   $Slot
        [UInt32]  $Count
        [Object] $Output
        OutputSection([UInt32]$Index,[String]$Name,[UInt32]$Slot)
        {
            $This.Index    = $Index
            $This.Name     = $Name
            $This.Slot     = $Slot
            $This.Clear()
        }
        Clear()
        {
            $This.Count    = 0
            $This.Output   = @( )
        }
        [Object] OutputProperty([UInt32]$Rank,[String]$Name,[Object]$Value)
        {
            Return [OutputProperty]::New($This.Name,$Rank,$Name,$Value)
        }
        Add([String]$Name,[Object]$Value)
        {
            $This.Output += $This.OutputProperty($This.Output.Count,$Name,$Value)
            $This.Count    = $This.Output.Count
        }
        [String] ToString()
        {
            Return "({0}) <FESystem.OutputSection>" -f $This.Count
        }
    }
```

```
                                                                                                   _____________________/
\__________________________________________________________________________________________________/ Class [OutputSection]
  Class [OutputFile] /----------------------------------------------------------------------------------------------------\
/--------------------
```

```
    # // ==================================================================
    # // | Orchestrates the necessary operations to format an output file |
    # // ==================================================================

    Class OutputFile
    {
        [Object] $Max
        [Object] $Output
        OutputFile([Object[]]$Object)
        {
            $This.Output = @( )
            $This.Max    = ($Object.Output.Name | Sort-Object Length)[-1]
            $Hash        = @{ }
            ForEach ($Item in $Object)
            {
                $This.Add($Hash,$This.Line())
                $This.Add($Hash,"[$($Item.Name)]")
                $This.Add($Hash,$This.Line())
                $This.Add($Hash," ")

                ForEach ($Prop in $Item.Output)
                {
                    If ($Prop.Value -match "^:$")
                    {
                        If ($Hash[$Hash.Count-1] -ne " ")
                        {
                            $This.Add($Hash," ")
                        }

                        $This.Add($Hash,$Prop.Name.PadRight($This.Max.Length," "))
                    }
                    Else
                    {
                        $This.Add($Hash,("{0} : {1}" -f $Prop.Name.PadRight($This.Max.Length," "), $Prop.Value))

                    }
                }

                $This.Add($Hash," ")
            }

            $This.Output = @($Hash[0..($Hash.Count-1)])
        }
        Add([Object]$Hash,[String]$Line)
        {
            $Hash.Add($Hash.Count,$Line)
        }
        [String] Line()
        {
            Return "=".PadLeft(120,"=")
        }
    }
```

```
                                                                                                                    ___________________/
\______________________________________________________________________________________________________________/ Class [OutputFile]
 Class [Snapshot] /‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

```
    # // ========================================================================================
    # // | Collection of properties such as hostname, basic network info, date/time, guid, etc. |
    # // ========================================================================================

    Class Snapshot
    {
        Hidden [UInt32]         $Mode
        Hidden [Object]           $OS
        Hidden [Object]           $CS
        [String]                $Start
        [String]         $ComputerName
        [String]                 $Name
        [String]          $DisplayName
        [String]                  $DNS
        [String]              $NetBIOS
        [String]             $Hostname
        [String]             $Username
        [Object]            $Principal
        [Bool]                $IsAdmin
        [String]              $Caption
        [Version]             $Version
        [UInt32]            $ReleaseID
        [UInt32]                $Build
        [String]                 $Code
        [String]          $Description
        [String]                  $SKU
        [String]              $Chassis
        [String]                 $Guid
        [UInt32]             $Complete
        [String]              $Elapsed
        Hidden [Object]      $Property
        Snapshot()
        {
            $This.Mode           = 0
            $Current             = $This.GetNow()
            $This.OS             = $This.GetOperatingSystem()
            $This.CS             = $This.GetComputerSystem()
            $This.Start          = $Current
            $This.ComputerName   = $This.GetMachineName()
            $This.Name           = $This.ComputerName.ToLower()
            $This.DisplayName    = "{0}-{1}" -f $Current.ToString("yyyy-MMdd-HHmmss"), $This.ComputerName
            $This.DNS            = @($Env:UserDNSDomain,"-")[!$env:USERDNSDOMAIN]
            $This.NetBIOS        = $This.GetUserDomainName().ToLower()
            $This.Hostname       = @($This.Name;"{0}.{1}" -f $This.Name, $This.DNS)
[$This.CS.PartOfDomain].ToLower()
            $This.Username       = $This.GetUserName()
            $This.Principal      = $This.GetPrincipal()
            $This.IsAdmin        = $This.Principal.IsInRole("Administrator") -or
$This.Principal.IsInRole("Administrators")
            $This.Caption        = $This.OS.Caption
            $This.GetFields()
            $This.Guid          = $This.NewGuid()

            $This.Property      = @( )

            ForEach ($Item in $This.PSObject.Properties)
            {
                $This.Property       += $This.SystemProperty($Item)
            }
        }
        Snapshot([Object]$Section)
        {
            $This.Mode = 1
            $This.Load($Section.Property)
        }
        Load([Object[]]$Pairs)
        {
            If ($This.Mode -ne 1)
            {
                Throw "Invalid mode"
            }

            ForEach ($Pair in $Pairs)
            {
                $This.$($Pair.Name) = $Pair.Value
            }
        }
        [Object] GetMachineName()
        {
            Return [Environment]::MachineName
        }
        [Object] GetUserDomainName()
        {
            Return [Environment]::UserDomainName
        }
        [Object] GetUsername()
        {
            Return [Environment]::UserName
        }
        [Object] GetNow()
        {
            Return [DateTime]::Now
        }
        [Object] NewGuid()
        {
            Return [Guid]::NewGuid()
        }
        [Object] GetPrincipal()
        {
            Return [Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()
        }
        [Object] GetComputerSystem()
        {
            Return Get-CimInstance Win32_ComputerSystem
        }
        [Object] GetOperatingSystem()
        {
            Return Get-CimInstance Win32_OperatingSystem
        }
        [Object] GetCurrentVersion()
        {
            Return Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion"
        }
        [Object] GetHost()
        {
            Return Get-Host
        }
        [String] GetEdition()
        {
            $Out = ("10240,Threshold 1,Release To Manufacturing;10586,Threshold 2,November {1};1439"+
            "3,{0} 1,Anniversary {1};15063,{0} 2,{2} {1};16299,{0} 3,Fall {2} {1};17134,{0} 4,Apri"+
            "l 2018 {1};17763,{0} 5,October 2018 {1};18362,19H1,May 2019 {1};18363,19H2,November 2"+
            "019 {1};19041,20H1,May 2020 {1};19042,20H2,October 2020 {1}") -f 'Redstone','Update',
            'Creators'

            $ID = Switch ($This.ReleaseID)
            {
                1507 {0} 1511 {1} 1607 {2} 1703 {3} 1709 {4} 1803 {5}
                1809 {6} 1903 {7} 1909 {8} 2004 {9} 2009 {10}
            }

            Return $Out.Split(";")[$Id]
        }
        [String] GetSku()
        {
            $Out = ("Undefined,Ultimate {0},Home Basic {0},Home Premium {0},{3} {0},Home Basic N {"+
            "0},Business {0},Standard {2} {0},Datacenter {2} {0},Small Business {2} {0},{3} {2} {0"+
            "},Starter {0},Datacenter {2} Core {0},Standard {2} Core {0},{3} {2} Core {0},{3} {2} "+
            "IA64 {0},Business N {0},Web {2} {0},Cluster {2} {0},Home {2} {0},Storage Express {2} "+
            "{0},Storage Standard {2} {0},Storage Workgroup {2} {0},Storage {3} {2} {0},{2} For Sm"+
            "all Business {0},Small Business {2} Premium {0},TBD,{1} {3},{1} Ultimate,Web {2} Core"+
            ",-,-,-,{2} Foundation,{1} Home {2},-,{1} {2} Standard No Hyper-V Full,{1} {2} Datacen"+
            "ter No Hyper-V Full,{1} {2} {3} No Hyper-V Full,{1} {2} Datacenter No Hyper-V Core,{1"+
            "} {2} Standard No Hyper-V Core,{1} {2} {3} No Hyper-V Core,Microsoft Hyper-V {2},Stor"+
            "age {2} Express Core,Storage {2} Standard Core,{2} Workgroup Core,Storage {2} {3} Cor"+
            "e,Starter N,Professional,Professional N,{1} Small Business {2} 2011 Essentials,-,-,-,"+
            "-,-,-,-,-,-,-,-,-,Small Business {2} Premium Core,{1} {2} Hyper Core V,-,-,-,-,-,-,-,"+
            "-,-,-,-,-,-,-,-,-,-,-,--,-,-,{1} Thin PC,-,{1} Embedded Industry,-,-,-,-,-,-,-,{1} RT"+
            ",-,-,Single Language N,{1} Home,-,{1} Professional with Media Center,{1} Mobile,-,-,-"+
            ",-,-,-,-,-,-,-,-,-,-,-,{1} Embedded Handheld,-,-,-,-,{1} IoT Core") -f "Edition",("Wind"+
            "ows"),"Server","Enterprise"

            Return $Out.Split(",")[$This.OS.OperatingSystemSku]
        }
        [String] GetChassis()
        {
            $Tag  = "N/A Desktop Mobile/Laptop Workstation {0} {0} Appliance {0} Max" -f "Server"
            Return $Tag.Split(" ")[$This.CS.PCSystemType]
        }
        GetFields()
        {
            $This.Version              = $This.GetHost().Version.ToString()
            $This.ReleaseID            = $This.GetCurrentVersion().ReleaseID

            $This.Build, $This.Code, $This.Description = $This.GetEdition() -Split ","

            $This.SKU                 = $This.GetSKU()
            $This.Chassis             = $This.GetChassis()
        }
        [Object] SystemProperty([Object]$Property)
        {
            Return [SystemProperty]::New(0,$This.Property.Count,"Snapshot",$Property.Name,$Property.Value)
        }
        MarkComplete()
        {
            $This.Complete     = 1
            $This.Elapsed      = [String][Timespan]([DateTime]::Now-[DateTime]$This.Start)
        }
        [UInt32] GetSlot()
        {
            Return 0
        }
        [String] ToString()
        {
            Return "{0}" -f $This.ComputerName
        }
    }
```

```
                                                                                                                        _________________/
\_____________________________________________________________________________________________________________________/ Class [Snapshot]
  Class [BiosInformation] /‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

```powershell
# // =======================================================
# // | Bios Information for the system this tool is run on |
# // =======================================================

Class BiosInformation
{
    Hidden [UInt32]     $Mode
    [String]            $Name
    [String]    $Manufacturer
    [String]    $SerialNumber
    [String]         $Version
    [String]     $ReleaseDate
    [Bool]     $SmBiosPresent
    [String]   $SmBiosVersion
    [String]     $SmBiosMajor
    [String]     $SmBiosMinor
    [String] $SystemBiosMajor
    [String] $SystemBiosMinor
    Hidden [Object] $Property
    BiosInformation()
    {
        $This.Mode            = 0
        $Bios                 = $This.CmdLet()
        $This.Name            = $Bios.Name
        $This.Manufacturer    = $Bios.Manufacturer
        $This.SerialNumber    = $Bios.SerialNumber
        $This.Version         = $Bios.Version
        $This.ReleaseDate     = $Bios.ReleaseDate
        $This.SmBiosPresent   = $Bios.SmBiosPresent
        $This.SmBiosVersion   = $Bios.SmBiosBiosVersion
        $This.SmBiosMajor     = $Bios.SmBiosMajorVersion
        $This.SmBiosMinor     = $Bios.SmBiosMinorVersion
        $This.SystemBiosMajor = $Bios.SystemBiosMajorVersion
        $This.SystemBIosMinor = $Bios.SystemBiosMinorVersion

        $This.Property        = @( )

        ForEach ($Item in $Bios.PSObject.Properties)
        {
            $This.Property   += $This.SystemProperty($Item)
        }
    }
    BiosInformation([Object]$Section)
    {
        $This.Mode = 1
        $This.Load($Section.Property)
    }
    Load([Object[]]$Pairs)
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        ForEach ($Pair in $Pairs)
        {
            $This.$($Pair.Name) = $Pair.Value
        }
    }
    [Object] CmdLet()
    {
        Return Get-CimInstance Win32_Bios
    }
    [Object] SystemProperty([Object]$Property)
    {
        Return [SystemProperty]::New(0,$This.Property.Count,"BiosInformation",$Property.Name,$Property.Value)
    }
    [UInt32] GetSlot()
    {
        Return 0
    }
```

```
        [String] ToString()
        {
            Return "{0} | {1}" -f $This.Manufacturer, $This.Name
        }
    }
```

```
                                                                                                            ________________________/
\_______________________________________________________________________________________________/ Class [BiosInformation]
  Class [ComputerSystem] /------------------------------------------------------------------------------------------------\
/------------------------
```

```
    # // ==================================================================
    # // | Computer system information for the system this tool is run on |
    # // ==================================================================

    Class ComputerSystem
    {
        Hidden [UInt32]     $Mode
        [String]    $Manufacturer
        [String]           $Model
        [String]         $Product
        [String]          $Serial
        [String]          $Memory
        [String]    $Architecture
        [String]            $UUID
        [String]         $Chassis
        [String]        $BiosUefi
        [Object]        $AssetTag
        Hidden [Object] $Property
        ComputerSystem()
        {
            $This.Mode         = 0

            $Computer          = @{

                System         = $This.GetComputerSystem()
                Product        = $This.GetComputerSystemProduct()
                Board          = $This.GetBaseBoard()
                Form           = $This.GetSystemEnclosure()
            }

            $This.Manufacturer = $Computer.System.Manufacturer
            $This.Model        = $Computer.System.Model
            $This.Memory       = "{0:n2} GB" -f ($Computer.System.TotalPhysicalMemory/1GB)
            $This.UUID         = $Computer.Product.UUID
            $This.Product      = $Computer.Product.Version
            $This.Serial       = $Computer.Board.SerialNumber -Replace "\.",""
            $This.BiosUefi     = $This.GetSecureBootUEFI()

            $This.AssetTag     = $Computer.Form.SMBIOSAssetTag.Trim()
            $This.Chassis      = Switch ([UInt32]$Computer.Form.ChassisTypes[0])
            {
                {$_ -in 8..12+14,18,21} {"Laptop"}
                {$_ -in 3..7+15,16}     {"Desktop"}
                {$_ -in 23}             {"Server"}
                {$_ -in 34..36}         {"Small Form Factor"}
                {$_ -in 30..32+13}      {"Tablet"}
            }

            $This.Architecture = @{

                          x86 = "x86"
                        AMD64 = "x64"

            }[[Environment]::GetEnvironmentVariable("Processor_Architecture")]

            $This.Property     = @( )

            ForEach ($Object in $Computer | % { $_.System, $_.Product, $_.Board, $_.Form})
            {
                ForEach ($Item in $Object.PSObject.Properties)
                {
                    $This.Property += $This.SystemProperty($Item)
                }
            }
        }
        ComputerSystem([Object]$Section)
        {
            $This.Mode = 1
            $This.Load($Section.Property)
        }
        Load([Object]$Pairs)
        {
            If ($This.Mode -ne 1)
            {
                Throw "Invalid mode"
            }

            ForEach ($Pair in $Pairs)
            {
                $This.$($Pair.Name) = $Pair.Value
            }
        }
        [Object] GetComputerSystem()
        {
            Return Get-CimInstance Win32_ComputerSystem
        }
        [Object] GetComputerSystemProduct()
        {
            Return Get-CimInstance Win32_ComputerSystemProduct
        }
        [Object] GetBaseboard()
        {
            Return Get-CimInstance Win32_Baseboard
        }
        [Object] GetSystemEnclosure()
        {
            Return Get-CimInstance Win32_SystemEnclosure
        }
        [String] GetSecureBootUEFI()
        {
            Try
            {
                Get-SecureBootUEFI -Name SetupMode
                Return "UEFI"
            }
            Catch
            {
                Return "BIOS"
            }
        }
        [Object] SystemProperty([Object]$Property)
        {
            Return [SystemProperty]::New(0,$This.Property.Count,"ComputerSystem",$Property.Name,$Property.Value)
        }
        [UInt32] GetSlot()
        {
            Return 0
        }
        [String] ToString()
        {
            Return "{0} | {1}" -f $This.Manufacturer, $This.Model
        }
    }
```

```
                                                                                                           ______________________/
\_________________________________________________________________________________________________________/ Class [ComputerSystem]
  Class [OperatingSystem] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
# // ===================================================================
# // | Operating system information for the system this tool is run on |
# // ===================================================================

Class OperatingSystem
{
    Hidden [UInt32]     $Mode
    Hidden [Object]       $Os
    [String]         $Caption
    [String]         $Version
    [String]           $Build
    [String]          $Serial
    [UInt32]        $Language
    [UInt32]         $Product
    [UInt32]            $Type
    Hidden [Object] $Property
    OperatingSystem()
    {
        $This.Mode          = 0

        $This.OS            = $This.CmdLet()

        $This.Caption       = $This.OS.Caption
        $This.Version       = $This.OS.Version
        $This.Build         = $This.OS.BuildNumber
        $This.Serial        = $This.OS.SerialNumber
        $This.Language      = $This.OS.OSLanguage
        $This.Product       = $This.OS.OSProductSuite
        $This.Type          = $This.OS.OSType

        $This.Property      = @( )

        ForEach ($Item in $This.OS.PSObject.Properties)
        {
            $This.AddProperty($Item)
        }
    }
    OperatingSystem([Object]$Section)
    {
        $This.Mode = 1
        $This.Load($Section.Property)
    }
    Load([Object[]]$Pairs)
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        ForEach ($Pair in $Pairs)
        {
            $This.$($Pair.Name) = $Pair.Value
        }
    }
    [Object] CmdLet()
    {
        Return Get-CimInstance Win32_OperatingSystem
    }
    [Object] SystemProperty([UInt32]$Index,[Object]$Property)
    {
        Return [SystemProperty]::New(0,$Index,"OperatingSystem",$Property.Name,$Property.Value)
    }
    AddProperty([Object]$Property)
    {
        $This.Property += $This.SystemProperty($This.Property.Count,$Property)
    }
    [UInt32] GetSlot()
    {
        Return 0
    }
    [String] ToString()
    {
        Return "{0} {1}-{2}" -f $This.Caption, $This.Version, $This.Build
    }
}
```

```
                                                                                                    ________________________/
\__________________________________________________________________________________________________/ Class [OperatingSystem]
  Class [HotFixItem] /--------------------------------------------------------------------------------------------------------\
/--------------------
```

```
    # // ================================================================
    # // | For enumerating installed Windows (Updates/Packages/HotFixes) |
    # // ================================================================

    Class HotFixItem
    {
        Hidden [UInt32]  $Index
        Hidden [Object] $HotFix
        Hidden [String] $Source
        [String]      $HotFixID
        [String]   $Description
        [String]   $InstalledBy
        [String]   $InstalledOn
        HotFixItem([UInt32]$Index,[Object]$HotFix)
        {
            $This.Index       = $Index
            $This.HotFix      = $HotFix
            $This.Source      = $HotFix.PSComputerName
            $This.Description = $HotFix.Description
            $This.HotFixID    = $HotFix.HotFixID
            $This.InstalledBy = $HotFix.InstalledBy
            $This.InstalledOn = ([DateTime]$HotFix.InstalledOn).ToString("MM/dd/yyyy")
        }
        HotFixItem([UInt32]$Index,[String]$Line,[Switch]$Flags)
        {
            $Trim             = $Line -Split "\|"
            $This.Index       = $Index
            $This.Source      = $Trim[0]
            $This.Description = $Trim[1]
            $This.HotFixId    = $Trim[2]
            $This.InstalledBy = $Trim[3]
            $This.InstalledOn = $Trim[4]
        }
        [String] Tag()
        {
            Return "HotFix{0}" -f $This.Index
        }
        [String] Value()
        {
            Return "{0}|{1}|{2}|{3}|{4}" -f $This.Source,
                                            $This.Description,
                                            $This.HotFixID,
                                            $This.InstalledBy,
                                            $This.InstalledOn
        }
        [UInt32] GetSlot()
        {
            Return 0
        }
        [String] ToString()
        {
            Return "<FESystem.HotFixItem>"
        }
    }
```

```
                                                                                                         __________________/
\_______________________________________________________________________________________________________/ Class [HotFixItem]
  Class [HotFixList] /--------------------------------------------------------------------------------------------------------\
/--------------------
```

```
# // ================================================================
# // | For enumerating installed Windows (Updates/Packages/HotFixes) |
# // ================================================================

Class HotFixList
{
    Hidden [UInt32] $Mode
    [String]        $Name
    [UInt32]       $Count
    [Object]      $Output
    HotFixList()
    {
        $This.Mode = 0
        $This.Name = "HotFix"
        $This.Refresh()
    }
    HotFixList([Object]$Section)
    {
        $This.Mode = 1
        $This.Name = "HotFix"
        $This.Clear()
        $This.Load($Section.Item)
    }
    Clear()
    {
        $This.Count  = 0
        $This.Output = @( )
    }
    Refresh()
    {
        If ($This.Mode -ne 0)
        {
            Throw "Invalid mode"
        }

        $This.Clear()

        ForEach ($HotFix in $This.CmdLet())
        {
            $This.Add($HotFix)
        }
    }
    Load([Object[]]$Items)
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        ForEach ($Item in $Items)
        {
            $This.Add($Item.Value,[Switch]$True)
        }
    }
    [Object[]] CmdLet()
    {
        Return Get-HotFix
    }
    [Object] HotFixItem([UInt32]$Index,[Object]$HotFix)
    {
        Return [HotFixItem]::New($Index,$Hotfix)
    }
    [Object] HotFixItem([UInt32]$Index,[String]$Line,[Switch]$Flags)
    {
        Return [HotFixItem]::New($Index,$Line,$Flags)
    }
    Add([Object]$Hotfix)
    {
        $This.Output += $This.HotFixItem($This.Output.Count,$HotFix)
        $This.Count   = $This.Output.Count
    }
```

```
        Add([String]$Line,[Switch]$Flags)
        {
            $This.Output += $This.HotFixItem($This.Output.Count,$Line,$Flags)
            $This.Count   = $This.Output.Count
        }
        [UInt32] GetSlot()
        {
            Return 1
        }
        [String] ToString()
        {
            Return "({0}) <FESystem.HotFixList>" -f $This.Count
        }
    }
```

```
                                                                                                                    __________________/
\___________________________________________________________________________________________________________/ Class [HotFixList]
  Class [WindowsOptionalStateSlot] /---------------------------------------------------------------------------------------\
/----------------------------------
```

```
    # // =====================================================
    # // | For enumerating Windows optional feature(s) state |
    # // =====================================================

    Enum WindowsOptionalStateType
    {
        Disabled
        DisabledWithPayloadRemoved
        Enabled
    }

    # // =====================================================
    # // | For enumerating Windows optional feature(s) state |
    # // =====================================================

    Class WindowsOptionalStateSlot
    {
        [UInt32] $Index
        [String] $Type
        [String] $Symbol
        [String] $Description
        WindowsOptionalStateSlot([String]$Type)
        {
            $This.Type   = [WindowsOptionalStateType]::$Type
            $This.Index  = [UInt32][WindowsOptionalStateType]::$Type
            $This.Symbol = @("[ ]","[!]","[+]")[$This.Index]
        }
        [String] ToString()
        {
            Return $This.Type
        }
    }
```

```
                                                                                                      ________________________________/
\_____________________________________________________________________________________________/ Class [WindowsOptionalStateSlot]
  Class [WindowsOptionalStateList] /---------------------------------------------------------------------------------------\
/----------------------------------
```

```
    # // =====================================================
    # // | For enumerating Windows optional feature(s) state |
    # // =====================================================

    Class WindowsOptionalStateList
    {
        [Object] $Output
        WindowsOptionalStateList()
        {
```

```
        $This.Output = @( )
        [System.Enum]::GetNames([WindowsOptionalStateType]) | % { $This.Add($_) }
    }
    Add([String]$Name)
    {
        $Item             = [WindowsOptionalStateSlot]::New($Name)
        $Item.Description = Switch ($Name)
        {
            Disabled                   { "Feature is disabled"                      }
            DisabledWithPayloadRemoved { "Feature is disabled, payload is removed" }
            Enabled                    { "Feature is enabled"                       }
        }
        $This.Output += $Item
    }
    [Object] Get([String]$Type)
    {
        Return $This.Output | ? Type -eq $Type
    }
}
```

```
                                                                                                    ________________________________/
\_____________________________________________________________________________________/ Class [WindowsOptionalStateList]
  Class [WindowsOptionalFeatureItem] /--------------------------------------------------------------------------------------------\
/------------------------------------
```

```
# // ================================================
# // | For enumerating Windows optional feature(s) |
# // ================================================

Class WindowsOptionalFeatureItem
{
    Hidden [UInt32]            $Index
    Hidden [Object]          $Feature
    [String]             $FeatureName
    [Object]                   $State
    Hidden [String]             $Path
    Hidden [UInt32]           $Online
    Hidden [String]          $WinPath
    Hidden [String]     $SysDrivePath
    Hidden [UInt32]    $RestartNeeded
    Hidden [String]          $LogPath
    Hidden [String] $ScratchDirectory
    Hidden [String]         $LogLevel
    WindowsOptionalFeatureItem([UInt32]$Index,[Object]$Feature)
    {
        $This.Index            = $Index
        $This.Feature          = $Feature
        $This.FeatureName      = $Feature.FeatureName
        $This.Path             = $Feature.Path
        $This.Online           = $Feature.Online
        $This.WinPath          = $Feature.WinPath
        $This.SysDrivePath     = $Feature.SysDrivePath
        $This.RestartNeeded    = $Feature.RestartNeeded
        $This.LogPath          = $Feature.LogPath
        $This.ScratchDirectory = $Feature.ScratchDirectory
        $This.LogLevel         = $Feature.LogLevel
    }
    WindowsOptionalFeatureItem([UInt32]$Index,[Object]$State,[String]$Feature)
    {
        $This.Index            = $Index
        $This.State            = $State
        $This.FeatureName      = $Feature
    }
    [String] StateLabel()
    {
        Return @("[ ]","[!]","[+]")[$This.State.Index]
    }
    [String] Tag()
    {
        Return "Feature{0}" -f $This.Index
    }
    [String] Value()
    {
        Return "{0} {1}" -f $This.StateLabel(), $This.FeatureName
    }
    [UInt32] GetSlot()
    {
        Return 0
    }
    [String] ToString()
    {
        Return "<FESystem.WindowsOptionalFeatureItem>"
    }
}
```

```
                                                                                      __________________________________/
\____________________________________________________________________________________/ Class [WindowsOptionalFeatureItem]
  Class [WindowsOptionalFeatureList] /---------------------------------------------------------------------------------\
/------------------------------------
```

```
    # // ===============================================
    # // | For enumerating Windows optional feature(s) |
    # // ===============================================

    Class WindowsOptionalFeatureList
    {
        Hidden [UInt32]  $Mode
        Hidden [Object] $State
        [String]         $Name
        [UInt32]        $Count
        [Object]       $Output
        WindowsOptionalFeatureList()
        {
            $This.Mode    = 0
            $This.Main()
            $This.Refresh()
        }
        WindowsOptionalFeatureList([Object]$Section)
        {
            $This.Mode    = 1
            $This.Main()
            $This.Clear()
            $This.Load($Section.Item)
        }
        Main()
        {
            $This.Name    = "Optional Features"
            $This.State   = $This.GetWindowsOptionalStateList()
        }
        Clear()
        {
            $This.Count  = 0
            $This.Output = @( )
        }
        Refresh()
        {
            If ($This.Mode -ne 0)
            {
                Throw "Invalid mode"
            }

            $This.Clear()

            ForEach ($Feature in $This.CmdLet())
            {
                $This.Add($Feature)
                $This.Output[-1].State = $This.State | ? Type -eq $This.Output[-1].Feature.State
            }
        }
        Load([Object[]]$Items)
        {
            If ($This.Mode -ne 1)
            {
                Throw "Invalid mode"
            }

            ForEach ($Item in $Items)
            {
                $xState  = $This.State | ? Symbol -match ([Regex]::Matches($Item.Value,"\[\W\]").Value)
                $This.Add($Item.Rank,$xState,$Item.Value.Substring(4))
            }
        }
        [Object] CmdLet()
        {
            Return Get-WindowsOptionalFeature -Online | Sort-Object FeatureName
        }
        [Object] WindowsOptionalFeatureItem([UInt32]$Index,[Object]$Feature)
        {
            Return [WindowsOptionalFeatureItem]::New($Index,$Feature)
        }
        [Object] WindowsOptionalFeatureItem([UInt32]$Index,[Object]$State,[String]$Name)
        {
            Return [WindowsOptionalFeatureItem]::New($Index,$State,$Name)
        }
        [Object] GetWindowsOptionalStateList()
        {
            Return [WindowsOptionalStateList]::New().Output
        }
        Add([Object]$Feature)
        {
            $This.Output += $This.WindowsOptionalFeatureItem($This.Output.Count,$Feature)
            $This.Count   = $This.Output.Count
        }
        Add([String]$Index,[Object]$State,[String]$Name)
        {
            $This.Output += $This.WindowsOptionalFeatureItem($Index,$State,$Name)
            $This.Count   = $This.Output.Count
        }
        [UInt32] GetSlot()
        {
            Return 1
        }
        [String] ToString()
        {
            Return "({0}) <FESystem.WindowsOptionalFeatureList>" -f $This.Count
        }
    }
```

```
                                                                                                    ___________________________________/
\__________________________________________________________________________________________________/ Class [WindowsOptionalFeatureList]
  Class [ApplicationItem] /--------------------------------------------------------------------------------------------------------------\
/-------------------------
```

```
    # // ================================================
    # // | For enumerating installed applications (Item) |
    # // ================================================

    Class ApplicationItem
    {
        Hidden [UInt32]           $Index
        Hidden [Object]     $Application
        [String]                   $Type
        [String]            $DisplayName
        [String]         $DisplayVersion
        Hidden [String]         $Version
        Hidden [Int32]         $NoRemove
        Hidden [String]      $ModifyPath
        Hidden [String] $UninstallString
        Hidden [String] $InstallLocation
        Hidden [String]     $DisplayIcon
        Hidden [Int32]         $NoRepair
        Hidden [String]       $Publisher
        Hidden [String]     $InstallDate
        Hidden [Int32]     $VersionMajor
        Hidden [Int32]     $VersionMinor
        ApplicationItem([UInt32]$Index,[Object]$App)
        {
            $This.Index             = $Index
            $This.Type              = @("MSI","WMI")[$App.UninstallString -imatch "msiexec"]
            $This.DisplayName       = @("-",$App.DisplayName)[!!$App.DisplayName]
            $This.DisplayVersion    = @("-",$App.DisplayVersion)[!!$App.DisplayVersion]
            $This.Version           = @("-",$App.Version)[!!$App.Version]
            $This.NoRemove          = $App.NoRemove
            $This.ModifyPath        = $App.ModifyPath
            $This.UninstallString   = $App.UninstallString
            $This.InstallLocation   = $App.InstallLocation
            $This.DisplayIcon       = $App.DisplayIcon
            $This.NoRepair          = $App.NoRepair
            $This.Publisher         = $App.Publisher
            $This.InstallDate       = $App.InstallDate
            $This.VersionMajor      = $App.VersionMajor
            $This.VersionMinor      = $App.VersionMinor
        }
        ApplicationItem([UInt32]$Index,[String]$Type,[String]$DisplayName,[String]$DisplayVersion)
        {
            $This.Index             = $Index
            $This.Type              = $Type
            $This.DisplayName       = $DisplayName
            $This.DisplayVersion    = $DisplayVersion
        }
        [String] Tag()
        {
            Return "Application{0}" -f $This.Index
        }
        [String] Value()
        {
            Return "{0}|{1}|{2}" -f $This.Type, $This.DisplayName, $This.DisplayVersion
        }
        [UInt32] GetSlot()
        {
            Return 0
        }
        [String] ToString()
        {
            Return "<FESystem.ApplicationItem>"
        }
    }
```

```
                                                                                                            ________________________/
\_______________________________________________________________________________________________________/ Class [ApplicationItem]
  Class [ApplicationList] /----------------------------------------------------------------------------------------------------------\
/-------------------------
```

```
    # // ======================================================
    # // | For enumerating installed applications (Container) |
    # // ======================================================

    Class ApplicationList
    {
        Hidden [UInt32] $Mode
        [String]        $Name
        [UInt32]       $Count
        [Object]      $Output
        ApplicationList()
        {
            $This.Mode   = 0
            $This.Main()
            $This.Refresh()
        }
```

```
ApplicationList([Object]$Section)
{
    $This.Mode   = 1
    $This.Main()
    $This.Load($Section.Item)
}
Main()
{
    $This.Name   = "Application"
    $This.Clear()
}
Clear()
{
    $This.Count  = 0
    $This.Output = @( )
}
Refresh()
{
    If ($This.Mode -ne 0)
    {
        Throw "Invalid mode"
    }

    $This.Clear()
    ForEach ($Application in $This.CmdLet())
    {
        $This.Add($Application)
    }
}
Load([Object[]]$Items)
{
    If ($This.Mode -ne 1)
    {
        Throw "Invalid mode"
    }

    ForEach ($Item in $Items)
    {
        $Trim           = $Item.Value -Split "\|"
        $Type           = $Trim[0]
        $DisplayName    = $Trim[1]
        $DisplayVersion = $Trim[2]

        $This.Add($Type,$DisplayName,$DisplayVersion)
    }
}
[String] GetArchitecture()
{
    Return [Environment]::GetEnvironmentVariable("Processor_Architecture")
}
[String[]] RegistryPath()
{
    $Item = "" , "\WOW6432Node" | % { "HKLM:\Software$_\Microsoft\Windows\CurrentVersion\Uninstall\*" }
    $Slot = Switch ($This.GetArchitecture())
    {
        AMD64   { 0,1 } Default { 0 }
    }

    Return $Item[$Slot]
}
[Object] CmdLet()
{
    Return $This.RegistryPath() | % { Get-ItemProperty $_ } | ? DisplayName | Sort-Object DisplayName
}
[Object] Application([UInt32]$Index,[Object]$Application)
{
    Return [ApplicationItem]::New($Index,$Application)
}
[Object] Application([UInt32]$Index,[String]$Type,[String]$DisplayName,[String]$DisplayVersion)
{
    Return [ApplicationItem]::New($Index,$Type,$DisplayName,$DisplayVersion)
}
```

```
        Add([Object]$Application)
        {
            $This.Output += $This.Application($This.Output.Count,$Application)
            $This.Count   = $This.Output.Count
        }
        Add([String]$Type,[String]$DisplayName,[String]$DisplayVersion)
        {
            $This.Output += $This.Application($This.Output.Count,$Type,$DisplayName,$DisplayVersion)
            $This.Count   = $This.Output.Count
        }
        [UInt32] GetSlot()
        {
            Return 1
        }
        [String] ToString()
        {
            Return "({0}) <FESystem.ApplicationList>" -f $This.Count
        }
    }
```

```
                                                                                                 _______________________/
\_______________________________________________________________________________________________/ Class [ApplicationList]
  Class [EventLogProviderItem] /----------------------------------------------------------------------------------------\
/------------------------------
```

```
    # // ==================================================================================
    # // | For enumerating all available Windows event log files (names/providers) (Item) |
    # // ==================================================================================

    Class EventLogProviderItem
    {
        Hidden [UInt32] $Index
        [String]          $Name
        [String]  $DisplayName
        EventLogProviderItem([UInt32]$Index,[String]$Name)
        {
            $This.Index       = $Index
            $This.Name        = "Provider$Index"
            $This.DisplayName = $Name
        }
        [String] Tag()
        {
            Return $This.Name
        }
        [String] Value()
        {
            Return $This.DisplayName
        }
        [UInt32] GetSlot()
        {
            Return 0
        }
        [String] ToString()
        {
            Return "<FESystem.EventLogProviderItem>"
        }
    }
```

```
                                                                                            ____________________________/
\__________________________________________________________________________________________/ Class [EventLogProviderItem]
  Class [EventLogProviderList] /----------------------------------------------------------------------------------------\
/------------------------------
```

```
    # // =======================================================================================
    # // | For enumerating all available Windows event log files (names/providers) (Container) |
    # // =======================================================================================
```

```powershell
Class EventLogProviderList
{
    Hidden [UInt32] $Mode
    [String]        $Name
    [UInt32]       $Count
    [Object]      $Output
    EventLogProviderList()
    {
        $This.Mode = 0
        $This.Main()
        $This.Refresh()
    }
    EventLogProviderList([Object]$Section)
    {
        $This.Mode = 1
        $This.Main()
        $This.Load($Section.Item)
    }
    Main()
    {
        $This.Name   = "Event Logs"
        $This.Clear()
    }
    Clear()
    {
        $This.Count  = 0
        $This.Output = @( )
    }
    Refresh()
    {
        If ($This.Mode -ne 0)
        {
            Throw "Invalid mode"
        }

        $This.Clear()

        ForEach ($Item in $This.CmdLet())
        {
            $This.Add($Item)
        }
    }
    Load([Object[]]$Items)
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        ForEach ($Item in $Items)
        {
            $This.Add($Item.Value)
        }
    }
    [Object] CmdLet()
    {
        Return Get-WinEvent -ListLog * | % LogName | Sort-Object
    }
    [Object] EventLogProviderItem([UInt32]$Index,[String]$Name)
    {
        Return [EventLogProviderItem]::New($Index,$Name)
    }
    Add([String]$Name)
    {
        $This.Output += $This.EventLogProviderItem($This.Output.Count,$Name)
        $This.Count   = $This.Output.Count
    }
    [UInt32] GetSlot()
    {
        Return 1
    }
    [String] ToString()
```

```
        {
            Return "({0}) <FESystem.EventLogProviderList>" -f $This.Count
        }
    }
```

```
                                                                                                    ____________________________/
\__________________________________________________________________________________________________/ Class [EventLogProviderList]
 Class [ScheduledTaskStateItem] /--------------------------------------------------------------------------------------------------\
/-------------------------------
```

```
    # // ========================================
    # // | For enumerating scheduled task states |
    # // ========================================

    Enum ScheduledTaskStateType
    {
        Disabled
        Ready
        Running
    }

    # // ====================================================================
    # // | For providing an index and description for a scheduled task state |
    # // ====================================================================

    Class ScheduledTaskStateItem
    {
        [UInt32]       $Index
        [String]        $Type
        [String] $Description
        ScheduledTaskStateItem([String]$Type)
        {
            $This.Type  = $Type
            $This.Index = [UInt32][ScheduledTaskStateType]::$Type
        }
        [String] ToString()
        {
            Return $This.Type
        }
    }
```

```
                                                                                                  ______________________________/
\________________________________________________________________________________________________/ Class [ScheduledTaskStateItem]
 Class [ScheduledTaskStateList] /--------------------------------------------------------------------------------------------------\
/-------------------------------
```

```
    # // ========================================
    # // | For enumerating scheduled task states |
    # // ========================================

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

            ForEach ($Name in [System.Enum]::GetNames([ScheduledTaskStateType]))
            {
                $This.Add($Name)
            }
        }
        Add([String]$Type)
        {
            $Item             = [ScheduledTaskStateItem]::New($Type)
            $Item.Description = Switch ($Type)
            {
                Disabled { "The scheduled task is currently disabled."        }
                Ready    { "The scheduled task is enabled, and ready to run." }
                Running  { "The scheduled task is currently running."         }
            }

            $This.Output     += $Item
        }
    }
```

```
                                                                                                   ______________________________/
\_________________________________________________________________________________________________/ Class [ScheduledTaskStateList]
 Class [ScheduledTaskItem] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
    # // ====================================
    # // | For enumerating scheduled task(s) |
    # // ====================================

    Class ScheduledTaskItem
    {
        Hidden [UInt32] $Index
        Hidden [Object]  $Task
        [String]         $Path
        [String]         $Name
        [Object]        $State
        ScheduledTaskItem([UInt32]$Index,[Object]$Task)
        {
            $This.Index = $Index
            $This.Task  = $Task
            $This.Path  = $Task.TaskPath
            $This.Name  = $Task.TaskName
        }
        ScheduledTaskItem([UInt32]$Index,[String]$Path,[String]$Name,[Object]$State)
        {
            $This.Index = $Index
            $This.Path  = $Path
            $This.Name  = $Name
            $This.State = $State
        }
        [String] Tag()
        {
            Return "Task{0}" -f $This.Index
        }
        [String] Value()
        {
            Return "{0}|{1}|{2}" -f $This.Path, $This.Name, $This.State.ToString()
        }
        [UInt32] GetSlot()
        {
            Return 0
        }
        [String] ToString()
        {
            Return "<FESystem.ScheduledTaskItem>"
        }
    }
```

```
                                                                                                        _________________________/
\______________________________________________________________________________________________________/ Class [ScheduledTaskItem]
 Class [ScheduledTaskList] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
    # // ====================================
    # // | For enumerating scheduled task(s) |
    # // ====================================

    Class ScheduledTaskList
    {
        Hidden [UInt32]  $Mode
        Hidden [Object] $State
        [String]         $Name
        [UInt32]        $Count
        [Object]       $Output
        ScheduledTaskList()
        {
            $This.Mode   = 0
            $This.Main()
            $This.Refresh()
        }
        ScheduledTaskList([Object]$Section)
        {
            $This.Mode   = 1
            $This.Main()
            $This.Load($Section.Item)
        }
        Main()
        {
            $This.Name   = "ScheduledTaskList"
            $This.State  = $This.ScheduledTaskStateList()
            $This.Clear()
        }
        Clear()
        {
            $This.Count  = 0
            $This.Output = @( )
        }
        Refresh()
        {
            If ($This.Mode -ne 0)
            {
                Throw "Invalid mode"
            }

            $This.Clear()

            ForEach ($Task in $This.CmdLet())
            {
                $This.Add($Task)
                $This.Output[-1].State = $This.State | ? Type -eq $Task.State
            }
        }
        Load([Object[]]$Items)
        {
            If ($This.Mode -ne 1)
            {
                Throw "Invalid mode"
            }

            ForEach ($Item in $Items)
            {
                $Trim   = $Item.Value -Split "\|"
                $Path   = $Trim[0]
                $xName  = $Trim[1]
                $xState = $This.State | ? Type -eq $Trim[2]

                $This.Add($Path,$xName,$xState)
            }
        }
        [Object] CmdLet()
        {
            Return Get-ScheduledTask
        }
        [Object] ScheduledTaskStateList()
        {
            Return [ScheduledTaskStateList]::New().Output
        }
        [Object] GetScheduledTaskItem([UInt32]$Index,[Object]$Task)
        {
            Return [ScheduledTaskItem]::New($Index,$Task)
        }
        [Object] GetScheduledTaskItem([UInt32]$Index,[String]$Path,[String]$Name,[Object]$State)
        {
            Return [ScheduledTaskItem]::New($Index,$Path,$Name,$State)
        }
        Add([Object]$Task)
        {
            $This.Output += $This.GetScheduledTaskItem($This.Output.Count,$Task)
            $This.Count    = $This.Output.Count
        }
        Add([String]$Path,[String]$Name,[Object]$State)
        {
            $This.Output += $This.GetScheduledTaskItem($This.Output.Count,$Path,$Name,$State)
            $This.Count    = $This.Output.Count
        }
        [UInt32] GetSlot()
        {
            Return 1
        }
        [String] ToString()
        {
            Return "({0}) <FESystem.ScheduledTaskList>" -f $This.count
        }
    }
```

```
                                                                                                ________________________/
\______________________________________________________________________________________________/ Class [ScheduledTaskList]
  Class [AppXItem] /-----------------------------------------------------------------------------------------------------\
/------------------
```

```
    # // =======================================================
    # // | # For enumerating AppX packages, like MS Edge, etc. |
    # // =======================================================

    Class AppXItem
    {
        Hidden [UInt32]            $Index
        Hidden [Object]             $AppX
        [String]             $DisplayName
        [Version]                $Version
        [String]             $PublisherID
        [String]             $PackageName
        Hidden [UInt32]     $MajorVersion
        Hidden [UInt32]     $MinorVersion
        Hidden [UInt32]            $Build
        Hidden [UInt32]         $Revision
        Hidden [UInt32]     $Architecture
        Hidden [String]       $ResourceID
        Hidden [String]  $InstallLocation
        Hidden [Object]          $Regions
        Hidden [String]             $Path
        Hidden [UInt32]           $Online
        Hidden [String]          $WinPath
        Hidden [string]     $SysDrivePath
        Hidden [UInt32]    $RestartNeeded
        Hidden [String]          $LogPath
        Hidden [String] $ScratchDirectory
        Hidden [String]         $LogLevel
        AppXItem([UInt32]$Index,[Object]$AppX)
        {
            $This.Index            = $Index
            $This.AppX             = $AppX
            $This.DisplayName      = $AppX.DisplayName
            $This.Version          = $AppX.Version
            $This.PublisherId      = $AppX.PublisherId
            $This.PackageName      = $AppX.PackageName
            $This.MajorVersion     = $AppX.MajorVersion
            $This.MinorVersion     = $AppX.MinorVersion
            $This.Build            = $AppX.Build
            $This.Revision         = $AppX.Revision
            $This.Architecture     = $AppX.Architecture
            $This.ResourceId       = $AppX.ResourceId
            $This.InstallLocation  = $AppX.InstallLocation
            $This.Regions          = $AppX.Regions
            $This.Path             = $AppX.Path
            $This.Online           = $AppX.Online
            $This.WinPath          = $AppX.WinPath
            $This.SysDrivePath     = $AppX.SysDrivePath
            $This.RestartNeeded    = $AppX.RestartNeeded
            $This.LogPath          = $AppX.LogPath
            $This.ScratchDirectory = $AppX.ScratchDirectory
            $This.LogLevel         = $AppX.LogLevel
        }
        AppXItem([UInt32]$Index,[String]$DisplayName,[String]$Version,[String]$PublisherID,[String]$PackageName)
        {
            $This.Index       = $Index
            $This.DisplayName = $DisplayName
            $This.Version     = $Version
            $This.PublisherID = $PublisherId
            $This.PackageName = $PackageName
        }
        [String] Tag()
        {
            Return "Application{0}" -f $This.Index
        }
        [String] Value()
        {
            Return "{0}|{1}|{2}|{3}" -f $This.DisplayName, $This.Version, $this.PublisherID, $This.PackageName
        }
        [UInt32] GetSlot()
        {
            Return 0
        }
        [String] ToString()
        {
            Return "<FESystem.AppXItem>"
        }
    }
```

```
                                                                                                                        ________________/
\______________________________________________________________________________________________________________________/ Class [AppXItem]
  Class [AppXList] /‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

```
    # // ====================================================
    # // | For enumerating AppX packages, like MS Edge, etc. |
    # // ====================================================

    Class AppXList
    {
        Hidden [UInt32] $Mode
        [String]        $Name
        [UInt32]       $Count
        [Object]      $Output
        AppXList()
        {
            $This.Mode = 0
            $This.Main()
            $This.Refresh()
        }
        AppXList([Object]$Section)
        {
            $This.Mode = 1
            $This.Main()
            $This.Load($Section.Item)
        }
        Main()
        {
            $This.Name = "AppXList"
            $This.Clear()
        }
        Clear()
        {
            $This.Count  = 0
            $This.Output = @( )
        }
        Refresh()
        {
            If ($This.Mode -ne 0)
            {
                Throw "Invalid mode"
            }

            $This.Clear()

            ForEach ($AppX in Get-AppxProvisionedPackage -Online)
            {
                $This.Add($AppX)
            }
        }
        Load([Object[]]$Items)
        {
            If ($This.Mode -ne 1)
            {
                Throw "Invalid mode"
            }

            ForEach ($Item in $Items)
            {
                $Trim        = $Item.Value -Split "\|"
                $DisplayName = $Trim[0]
                $Version     = $Trim[1]
                $PublisherId = $Trim[2]
                $PackageName = $Trim[3]

                $This.Add($DisplayName,$Version,$PublisherId,$PackageName)
            }
        }
        [Object] AppXItem([UInt32]$Index,[Object]$AppX)
        {
            Return [AppXItem]::New($Index,$AppX)
        }
        [Object] AppXItem([UInt32]$Index,[String]$Display,[String]$Version,[String]$PubID,[String]$Pkg)
        {
            Return [AppXItem]::New($Index,$Display,$Version,$PubID,$Pkg)
        }
        Add([Object]$AppX)
        {
            $This.Output += $This.AppXItem($This.Output.Count,$AppX)
            $This.Count   = $This.Output.Count
        }
        Add([String]$DisplayName,[String]$Version,[String]$PublisherID,[String]$PackageName)
        {
            $This.Output += $This.AppXItem($This.Output.Count,$DisplayName,$Version,$PublisherID,$PackageName)
            $This.Count   = $This.Output.Count
        }
        [UInt32] GetSlot()
        {
            Return 1
        }
        [String] ToString()
        {
            Return "({0}) <FESystem.AppXList>" -f $This.Count
        }
    }
```

```
                                                                                                        ________________/
\_______________________________________________________________________________________________________/ Class [AppXList]
  Class [ProcessorItem] /-----------------------------------------------------------------------------------------------\
/-----------------------
```

```
# // ============================================================
# // | Processor information for the system this tool is run on |
# // ============================================================

Class ProcessorItem
{
    Hidden [UInt32]     $Mode
    [UInt32]            $Rank
    [String]    $Manufacturer
    [String]            $Name
    [String]         $Caption
    [UInt32]           $Cores
    [UInt32]            $Used
    [UInt32]         $Logical
    [UInt32]         $Threads
    [String]     $ProcessorId
    [String]        $DeviceId
    [UInt32]           $Speed
    Hidden [Object] $Property
    ProcessorItem([UInt32]$Rank,[Object]$CPU)
    {
        $This.Mode         = 0
        $This.Rank         = $Rank
        $This.Manufacturer = Switch -Regex ($CPU.Manufacturer)
                             {
                                 Intel   {           "Intel" }
                                 Amd     {             "AMD" }
                                 Default { $Cpu.Manufacturer }
                             }
        $This.Name         = $CPU.Name -Replace "\s+"," "
        $This.Caption      = $CPU.Caption
        $This.Cores        = $CPU.NumberOfCores
        $This.Used         = $CPU.NumberOfEnabledCore
        $This.Logical      = $CPU.NumberOfLogicalProcessors
        $This.Threads      = $CPU.ThreadCount
        $This.ProcessorID  = $CPU.ProcessorId
        $This.DeviceID     = $CPU.DeviceID
        $This.Speed        = $CPU.MaxClockSpeed

        $This.Property     = @( )

        ForEach ($Item in $CPU.PSObject.Properties)
        {
            $This.Property += $This.SystemProperty($Item)
        }
    }
    ProcessorItem([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
    {
        $This.Mode         = 1
        $This.Rank         = $Rank
        $This.Load($Pairs)
    }
    Load([Object[]]$Pairs)
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        ForEach ($Pair in $Pairs)
        {
            $This.$($Pair.Name) = $Pair.Value
        }
    }
    [Object] SystemProperty([Object]$Property)
    {
        Return [SystemProperty]::New(0,
                                     $This.Property.Count,
                                     "Processor$($This.Rank)",
                                     $Property.Name,
                                     $Property.Value)
    }
    [UInt32] GetSlot()
    {
        Return 0
    }
    [String] ToString()
    {
        Return $This.Name
    }
}
```

```
                                                                                                                    ____________________/
\__________________________________________________________________________________________________________________/ Class [ProcessorItem]
  Class [ProcessorList] /‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

```
    # // ==========================================
    # // | Processor container, handles 1 or more |
    # // ==========================================

    Class ProcessorList
    {
        Hidden [UInt32] $Mode
        [Object]        $Name
        [Object]       $Count
        [Object]      $Output
        ProcessorList()
        {
            $This.Mode = 0
            $This.Main()
            $This.Refresh()
        }
        ProcessorList([Object]$Section)
        {
            $This.Mode = 1
            $This.Main()
            $This.Load($Section.Item)
        }
        Main()
        {
            $This.Name     = "Processor(s)"
            $This.Clear()
        }
        Clear()
        {
            $This.Output   = @( )
            $This.Count    = 0
        }
        Refresh()
        {
            If ($This.Mode -eq 1)
            {
                Throw "Invalid mode"
            }

            $This.Clear()

            ForEach ($Processor in $This.CmdLet())
            {
                $This.Add($This.Output.Count,$Processor)
            }
        }
        Load([Object[]]$Items)
        {
            If ($This.Mode -eq 0)
            {
                Throw "Invalid mode"
            }

            ForEach ($Item in $Items)
            {
                $This.Add($Item.Rank,$Item.Property,[Switch]$True)
            }
        }
        [Object] CmdLet()
        {
            Return Get-CimInstance Win32_Processor
        }
        [Object] ProcessorItem([UInt32]$Rank,[Object]$Processor)
        {
            Return [ProcessorItem]::New($Rank,$Processor)
        }
        [Object] ProcessorItem([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
        {
            Return [ProcessorItem]::New($Rank,$Pairs,$Flags)
        }
        Add([UInt32]$Rank,[Object]$Processor)
        {
            $This.Output  += $This.ProcessorItem($Rank,$Processor)
            $This.Count    = $This.Output.Count
        }
        Add([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
        {
            $This.Output  += $This.ProcessorItem($Rank,$Pairs,$Flags)
            $This.Count    = $This.Output.Count
        }
        [UInt32] GetSlot()
        {
            Return 1
        }
        [String] ToString()
        {
            Return "({0}) <FESystem.ProcessorList>" -f $This.Count
        }
    }
```

```
                                                                                                    _____________________/
\__________________________________________________________________________________________________/ Class [ProcessorList]
  Class [Size] /--------------------------------------------------------------------------------------------------------\
/--------------
```

```
    # // ======================================================================
    # // | Used to convert the byte size of a drive or partition into a string |
    # // ======================================================================

    Class Size
    {
        [UInt64] $Bytes
        [String] $String
        Size([UInt64]$Bytes)
        {
            $This.Bytes  = $Bytes
            $This.String = $This.GetSize($Bytes)
        }
        [String] GetSize([Int64]$Size)
        {
            Return @( Switch ($Size)
            {
                {$_ -lt 1KB}                 {     "{0} B" -f $Size }
                {$_ -ge 1KB -and $_ -lt 1MB} { "{0:n2} KB" -f ($Size/1KB) }
                {$_ -ge 1MB -and $_ -lt 1GB} { "{0:n2} MB" -f ($Size/1MB) }
                {$_ -ge 1GB -and $_ -lt 1TB} { "{0:n2} GB" -f ($Size/1GB) }
                {$_ -ge 1TB}                 { "{0:n2} TB" -f ($Size/1TB) }
            })
        }
        [String] ToString()
        {
            Return $This.String
        }
    }
```

```
                                                                                                                           ___________/
\_______________________________________________________________________________________________________________________/ Class [Size]
  Class [PartitionItem] /----------------------------------------------------------------------------------------------------------\
/-----------------------
```

```
# // ==================================================================
# // | Drive/partition information for the system this tool is run on |
# // ==================================================================

Class PartitionItem
{
    Hidden [UInt32]      $Mode
    Hidden [String]     $Label
    [UInt32]             $Rank
    [String]             $Type
    [String]             $Name
    [Object]             $Size
    [UInt32]             $Boot
    [UInt32]          $Primary
    [UInt32]             $Disk
    [UInt32]        $Partition
    Hidden [Object] $Property
    PartitionItem([UInt32]$Rank,[Object]$Partition)
    {
        $This.Label      = $Partition.Name -Replace "( |#)", "" -Replace ",","."
        $This.Rank       = $Rank
        $This.Type       = $Partition.Type
        $This.Name       = $Partition.Name
        $This.Size       = $This.GetSize($Partition.Size)
        $This.Boot       = $Partition.BootPartition
        $This.Primary    = $Partition.PrimaryPartition
        $This.Disk       = $Partition.DiskIndex
        $This.Partition  = $Partition.Index

        $This.Property   = @( )

        ForEach ($Item in $Partition.PSObject.Properties)
        {
            $This.Property += $This.SystemProperty($Item)
        }
    }
    PartitionItem([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
    {
        $This.Mode = 1
        $This.Rank = $Rank
        $This.Load($Pairs)
    }
    Load([Object[]]$Pairs)
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        ForEach ($Pair in $Pairs)
        {
            $This.$($Pair.Name) = $Pair.Value
        }
    }
    [Object] SystemProperty([Object]$Property)
    {
        Return [SystemProperty]::New(0,
                                     $This.Property.Count,
                                     "Disk$($This.Disk).Partition$($This.Partition)",
                                     $Property.Name,
                                     $Property.Value)
    }
    [Object] GetSize([UInt64]$Bytes)
    {
        Return [Size]::New($Bytes)
    }
    [UInt32] GetSlot()
    {
        Return 0
    }
    [String] ToString()
    {
        Return "[{0}/{1}]" -f $This.Name, $This.Size
    }
}
```

```
                                                                                                  ______________________/
\________________________________________________________________________________________________/ Class [PartitionItem]
  Class [PartitionList] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
    # // ================================================================
    # // | Specifically for single/multiple partitions on a given drive |
    # // ================================================================

    Class PartitionList
    {
        Hidden [UInt32] $Mode
        [String]        $Name
        [UInt32]       $Count
        [Object]      $Output
        PartitionList()
        {
            $This.Mode = 0
            $This.Main()
        }
        PartitionList([Object[]]$Section)
        {
            $This.Mode = 1
            $This.Main()

            ForEach ($Partition in $Section | ? Name -match Partition)
            {
                $Rank = $Partition.Name -Replace "Disk\d+\.Partition", ""
                $This.Add($Rank,$Partition.Property,[Switch]$True)
            }
        }
        Main()
        {
            $This.Name = "Partition(s)"
            $This.Clear()
        }
        Clear()
        {
            $This.Count   = 0
            $this.Output  = @( )
        }
        [Object] PartitionItem([UInt32]$Rank,[Object]$Partition)
        {
            Return [PartitionItem]::New($Rank,$Partition)
        }
        [Object] PartitionItem([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
        {
            Return [PartitionItem]::New($Rank,$Pairs,$Flags)
        }
        Add([UInt32]$Rank,[Object]$Partition)
        {
            $This.Output += $This.PartitionItem($Rank,$Partition)
            $This.Count   = $This.Output.Count
        }
        Add([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
        {
            $This.Output += $This.PartitionItem($Rank,$Pairs,[Switch]$Flags)
            $This.Count   = $This.Output.Count
        }
        [UInt32] GetSlot()
        {
            Return 1
        }
        [String] ToString()
        {
            Return "({0}) {1}" -f $This.Count, (($This.Output | % ToString) -join ", ")
        }
    }
```

```
                                                                                                    _____________________/
\__________________________________________________________________________________________________/ Class [PartitionList]
  Class [VolumeItem] /‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

```
# // ===========================================================================================
# // | Specifically for a single volume on a given drive, meant for injecting with a partition |
# // ===========================================================================================

Class VolumeItem
{
    Hidden [String]    $Label
    [UInt32]            $Rank
    [String]         $DriveID
    [String]     $Description
    [String]      $Filesystem
    [Object]       $Partition
    [String]      $VolumeName
    [String]    $VolumeSerial
    [Object]            $Size
    [Object]       $Freespace
    [Object]            $Used
    Hidden [Object] $Property
    VolumeItem([UInt32]$Rank,[String]$Partition,[Object]$Drive)
    {
        $This.Label              = "{0}.Volume$Rank" -f ($Partition -Split ",")[0] -Replace "( |#)",""
        $This.Rank               = $Rank
        $This.DriveID            = $Drive.Name
        $This.Description        = $Drive.Description
        $This.Filesystem         = $Drive.Filesystem
        $This.Partition          = $Partition
        $This.VolumeName         = $Drive.VolumeName
        $This.VolumeSerial       = $Drive.VolumeSerialNumber
        $This.Size               = $This.GetSize($Drive.Size)
        $This.Freespace          = $This.GetSize($Drive.Freespace)
        $This.Used               = $This.GetSize(($This.Size.Bytes - $This.Freespace.Bytes))

        $This.Property           = @( )

        ForEach ($Item in $This.PSObject.Properties)
        {
            $This.Property += $This.SystemProperty($Item)
        }
    }
    VolumeItem([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
    {
        $This.Rank               = $Rank

        ForEach ($Pair in $Pairs)
        {
            $This.$($Pair.Name) = $Pair.Value
        }
    }
    [Object] SystemProperty([Object]$Property)
    {
        Return [SystemProperty]::New(0,$This.Property.Count,$This.Label,$Property.Name,$Property.Value)
    }
    [Object] GetSize([UInt64]$Bytes)
```

```
        {
            Return [Size]::New($Bytes)
        }
        [UInt32] GetSlot()
        {
            Return 0
        }
        [String] ToString()
        {
            Return "[{0}\ {1}]" -f $This.DriveID, $This.Size
        }
    }
```

```
                                                                                                      ___________________/
\____________________________________________________________________________________________________/ Class [VolumeItem]
  Class [VolumeList] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
    # // ==============================================================
    # // | Specifically for single/multiple volumes on a given drive |
    # // ==============================================================

    Class VolumeList
    {
        Hidden [UInt32] $Mode
        [String]        $Name
        [UInt32]       $Count
        [Object]      $Output
        VolumeList()
        {
            $This.Mode = 0
            $This.Main()
        }
        VolumeList([Object[]]$Section)
        {
            $This.Mode = 1
            $This.Main()

            ForEach ($Volume in $Section | ? Name -match Volume)
            {
                $Rank = $Volume.Name -Replace "Disk\d+\.Volume", ""
                $This.Add($Rank,$Volume.Property,[Switch]$True)
            }
        }
        Main()
        {
            $This.Name = "Volumes"
            $This.Clear()
        }
        Clear()
        {
            $This.Count   = 0
            $This.Output  = @( )
        }
        [Object] VolumeItem([UInt32]$Rank,[String]$Partition,[Object]$Drive)
        {
            Return [VolumeItem]::New($Rank,$Partition,$Drive)
        }
        [Object] VolumeItem([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
        {
            Return [VolumeItem]::New($Rank,$Pairs,$Flags)
        }
        Add([UInt32]$Rank,[String]$Partition,[Object]$Drive)
        {
            $This.Output += $This.VolumeItem($Rank,$Partition,$Drive)
            $This.Count   = $This.Output.Count
        }
        Add([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
        {
            $This.Output += $This.VolumeItem($Rank,$Pairs,$Flags)
            $This.Count   = $This.Output.Count
        }
        [UInt32] GetSlot()
        {
            Return 1
        }
        [String] ToString()
        {
            Return "({0}) {1}" -f $This.Count, (($This.Output | % ToString) -join ", ")
        }
    }
```

```
                                                                                                      __________________/
\____________________________________________________________________________________________________/ Class [VolumeList]
  Class [DiskItem] /‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

```
# // ========================================
# // | Extended information for hard drives |
# // ========================================

Class DiskItem
{
    Hidden [UInt32]       $Mode
    Hidden [UInt32]       $Rank
    [UInt32]             $Index
    [String]              $Disk
    [String]             $Model
    [String]            $Serial
    [String]    $PartitionStyle
    [String]  $ProvisioningType
    [String] $OperationalStatus
    [String]      $HealthStatus
    [String]           $BusType
    [String]          $UniqueId
    [String]          $Location
    [Object]         $Partition
    [Object]            $Volume
    DiskItem([UInt32]$Rank,[Object]$Disk)
    {
        $This.Mode     = 0
        $This.Rank     = $Rank
        $This.Index    = $Disk.Index
        $This.Disk     = $Disk.DeviceId

        $This.Init()

        $This.Refresh()
    }
    DiskItem([UInt32]$Rank,[Object[]]$Section,[Switch]$Flags)
    {
        $This.Mode    = 1
        $This.Rank    = $Rank

        # [Properties]
        ForEach ($Pair in $Section | ? Name -match ^Disk\d+$ | % Property)
        {
            $This.$($Pair.Name) = $Pair.Value
        }

        # [Partition(s) + Volume(s)]
        $This.Init($Section)
    }
    [Object] CmdLet([String]$Name)
    {
        If ($This.Mode -ne 0)
        {
            Throw "Invalid mode"
        }
        $Item = Switch -Regex ($Name)
        {
            MsftDisk
            {
                Get-CimInstance MSFT_Disk -N Root/Microsoft/Windows/Storage | ? Number -eq $This.Index
            }
            DiskPartition
            {
                Get-CimInstance Win32_DiskPartition | ? DiskIndex -eq $This.Index
            }
            LogicalDisk
            {
                Get-CimInstance Win32_LogicalDisk | ? DriveType -eq 3
            }
            LogicalDiskToPartition
            {
                Get-CimInstance Win32_LogicalDiskToPartition
            }
        }

        Return $Item
    }
    [Object] GetPartitionList()
    {
        Return [PartitionList]::New()
    }
    [Object] GetPartitionList([Object[]]$Section,[Switch]$Flags)
    {
        Return [PartitionList]::New($Section,$Flags)
    }
    [Object] GetVolumeList()
    {
        Return [VolumeList]::New()
    }
    [Object] GetVolumeList([Object[]]$Section,[Switch]$Flags)
    {
        Return [VolumeList]::New($Section,$Flags)
    }
    Init()
    {
        $This.Partition = $This.GetPartitionList()
        $This.Volume    = $This.GetVolumeList()
    }
    Init([Object[]]$Section)
    {
        $This.Partition = $This.GetPartitionList($Section,[Switch]$True)
        $This.Volume    = $This.GetVolumeList($Section,[Switch]$True)
    }
    Refresh()
    {
        If ($This.Mode -ne 0)
        {
            Throw "Invalid mode"
        }

        $MSFTDISK                = $This.CmdLet("MsftDisk")
        If (!$MSFTDISK)
        {
            Throw "Unable to set the drive data"
        }

        $This.Model              = $MSFTDISK.Model
        $This.Serial             = $MSFTDISK | ? SerialNumber | % { $_.SerialNumber.TrimStart(" ") }
        $This.PartitionStyle     = $MSFTDISK.PartitionStyle
        $This.ProvisioningType   = $MSFTDISK.ProvisioningType
        $This.OperationalStatus  = $MSFTDISK.OperationalStatus
        $This.HealthStatus       = $MSFTDISK.HealthStatus
        $This.BusType            = $MSFTDISK.BusType
        $This.UniqueId           = $MSFTDISK.UniqueId
        $This.Location           = $MSFTDISK.Location

        $DiskPartition           = $This.CmdLet("DiskPartition")
        $LogicalDisk            = $This.CmdLet("LogicalDisk")
        $LogicalPart            = $This.CmdLet("LogicalDiskToPartition")

        Switch ($DiskPartition.Count)
        {
            0
            {
                Write-Warning "[Disk:($($This.Rank))] [!] No disk partitions detected [!]"
            }
            Default
            {
                ForEach ($Item in $DiskPartition)
                {
                    $This.Partition.Add($This.Partition.Output.Count,$Item)
                }
            }
        }

        Switch ($LogicalDisk.Count)
        {
            0
            {
                Write-Warning "[Disk: $($This.Rank)] [!] No disk volumes detected [!]"
            }
            Default
            {
                ForEach ($Logical in $LogicalPart | ? { $_.Antecedent.DeviceID -in $DiskPartition.Name})
                {
                    $Part = $DiskPartition | ? Name     -eq $Logical.Antecedent.DeviceID
                    $Item = $LogicalDisk   | ? DeviceID -eq $Logical.Dependent.DeviceID
                    If ($Part -and $Item)
                    {
                        $This.Volume.Add($This.Volume.Output.Count,$Part.Name,$Item)
                    }
                }
            }
        }
    }
    [UInt32] GetSlot()
    {
        Return 2
    }
    [String] ToString()
    {
        Return "{0}({1})" -f $This.Model, $This.Rank
    }
}
```

```
                                                                                                        ________________/
\______________________________________________________________________________________________________/ Class [DiskItem]
  Class [DiskList] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
# // =====================================================================================
# // | Drive/file formatting information (container), for the system this tool is run on |
# // =====================================================================================

Class DiskList
{
    Hidden [UInt32] $Mode
    [Object]        $Name
    [Object]       $Count
    [Object]      $Output
    DiskList()
    {
        $This.Mode = 0
        $This.Main()
    }
    DiskList([Object]$Section)
    {
        $This.Mode = 1
        $This.Main()
        $This.Load($Section.Item)
    }
    Main()
    {
        $This.Name      = "Disk(s)"
        $This.Clear()
    }
    Clear()
    {
        $This.Count     = 0
        $This.Output    = @( )
    }
    Refresh()
    {
        If ($This.Mode -eq 1)
        {
            Throw "Invalid mode"
        }

        $This.Clear()

        ForEach ($Disk in Get-CimInstance Win32_DiskDrive | ? MediaType -match Fixed)
        {
            $This.Add($Disk)
        }
    }
    Load([Object[]]$Items)
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        # Queues the number of unique disks
        ForEach ($DiskName in $Items | ? Name -match ^Disk\d+$ | % Name)
        {
            # Selects ALL of the items belonging to the current disk name
            $Current    = $Items | ? Name -match $DiskName
            $Rank       = $Diskname -Replace "Disk",""

            $This.Add($Rank,$Current)
        }
    }
    [Object] DiskItem([UInt32]$Rank,[Object]$Disk)
    {
        Return [DiskItem]::New($Rank,$Disk)
    }
    [Object] DiskItem([UInt32]$Rank,[Object[]]$Items,[Switch]$Flags)
    {
        Return [DiskItem]::New($Rank,$Items,$Flags)
    }
    Add([Object]$Disk)
    {
        $This.Output += $This.DiskItem($This.Output.Count,$Disk)
        $This.Count    = $This.Output.Count
    }
    Add([UInt32]$Rank,[Object[]]$Items)
    {
        $This.Output += $This.DiskItem($Rank,$Items,[Switch]$True)
        $This.Count    = $This.Output.Count
    }
    [UInt32] GetSlot()
    {
        Return 2
    }
    [String] ToString()
    {
        Return "({0}) <FESystem.DiskList>" -f $This.Count
    }
}
```

```
                                                                                                        _________________/
\______________________________________________________________________________________________________/ Class [DiskList]
  Class [NetworkItem] /-------------------------------------------------------------------------------------------------\
/---------------------
```

```
# // ==================================================
# // | Network adapter information (Online + Offline) |
# // ==================================================

Class NetworkItem
{
    [UInt32]       $Rank
    [String]       $Name
    [UInt32]     $Status
    [String]  $IPAddress
    [String] $SubnetMask
    [String]    $Gateway
    [String]  $DnsServer
    [String] $DhcpServer
    [String] $MacAddress
    NetworkItem([UInt32]$Rank,[Object]$If)
    {
        $This.Rank            = $Rank
        $This.Name            = $If.Description
        $This.Status          = [UInt32]$If.IPEnabled
        Switch ($This.Status)
        {
            0
            {
                $This.IPAddress   = "-"
                $This.SubnetMask  = "-"
                $This.Gateway     = "-"
                $This.DnsServer   = "-"
                $This.DhcpServer  = "-"
            }
            1
            {
                $This.IPAddress   = $If.IPAddress             | ? {$_ -match "(\d+\.){3}\d+"}
                $This.SubnetMask  = $If.IPSubnet              | ? {$_ -match "(\d+\.){3}\d+"}
                If ($If.DefaultIPGateway)
                {
                    $This.Gateway = $If.DefaultIPGateway      | ? {$_ -match "(\d+\.){3}\d+"}
                }
                $This.DnsServer   = ($If.DnsServerSearchOrder | ? {$_ -match "(\d+\.){3}\d+"}) -join ", "
                $This.DhcpServer  = $If.DhcpServer            | ? {$_ -match "(\d+\.){3}\d+"}
            }
        }
        $This.MacAddress      = ("-",$If.MacAddress)[!!$If.MacAddress]
    }
    NetworkItem([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
    {
        $This.Rank              = $Rank

        ForEach ($Pair in $Pairs)
        {
            $This.$($Pair.Name) = $Pair.Value
        }
    }
    [UInt32] GetSlot()
    {
        Return 0
    }
    [String] ToString()
    {
        Return $This.Name
    }
}
```

```
                                                                                                                   ___________________/
\_________________________________________________________________________________________________________________/ Class [NetworkItem]
  Class [NetworkList] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
# // ====================================
# // | Network adapter container object |
# // ====================================

Class NetworkList
{
    Hidden [UInt32] $Mode
    [Object]        $Name
    [Object]       $Count
    [Object]      $Output
    NetworkList()
    {
        $This.Mode = 0
        $This.Main()
    }
    NetworkList([Object]$Section)
    {
        $This.Mode = 1
        $This.Main()
        $This.Load($Section.Item)
    }
    Main()
    {
        $This.Name       = "Network(s)"
        $This.Clear()
    }
    Clear()
    {
        $This.Output      = @( )
        $This.Count       = 0
    }
    [Object] CmdLet()
    {
        Return Get-CimInstance Win32_NetworkAdapterConfiguration
    }
    Refresh()
    {
        If ($This.Mode -ne 0)
        {
            Throw "Invalid mode"
        }

        $This.Clear()

        ForEach ($Network in $This.CmdLet())
        {
            $This.Add($Network)
        }
    }
    Load([Object[]]$Items)
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        ForEach ($Item in $Items)
        {
            $Rank = $Item.Name -Replace "Network",""
            $This.Add($Rank,$Item.Property,[Switch]$True)
        }
    }
    [Object] NetworkItem([UInt32]$Index,[Object]$Network)
    {
        Return [NetworkItem]::New($Index,$Network)
    }
    [Object] NetworkItem([UInt32]$Index,[Object[]]$Pairs,[Object]$Flags)
    {
        Return [NetworkItem]::New($Index,$Pairs,$Flags)
    }
    Add([Object]$Network)
    {
        $This.Output     += $This.NetworkItem($This.Output.Count,$Network)
        $This.Count       = $This.Output.Count
    }
    Add([UInt32]$Rank,[Object[]]$Pairs,[Switch]$Flags)
    {
        $This.Output     += $This.NetworkItem($Rank,$Pairs,$Flags)
        $This.Count       = $This.Output.Count
    }
    [UInt32] GetSlot()
    {
        Return 1
    }
    [String] ToString()
    {
        Return "({0}) <FESystem.NetworkList>" -f $This.Count
    }
}
```

```
                                                                                                     ___________________/
\___________________________________________________________________________________________________/ Class [NetworkList]
  Class [System] /------------------------------------------------------------------------------------------------------\
/----------------
```

```
# // =====================================================
# // | System snapshot, the primary focus of the utility |
# // =====================================================

Class System
{
    Hidden [UInt32]     $Mode
    [Object]        $Snapshot
    [Object] $BiosInformation
    [Object]  $ComputerSystem
    [Object] $OperatingSystem
    [Object]          $HotFix
    [Object]         $Feature
    [Object]     $Application
    [Object]           $Event
    [Object]            $Task
    [Object]            $AppX
    [Object]       $Processor
    [Object]            $Disk
    [Object]         $Network
    System()
    {
        $This.Mode              = 0
        $This.Snapshot          = $This.New(0)
        If (!$This.Snapshot.IsAdmin)
        {
            Throw "Must run as administrator"
        }

        $This.BiosInformation  = $This.New(1)
        $This.ComputerSystem   = $This.New(2)
        $This.OperatingSystem  = $This.New(3)
        $This.HotFix           = $This.New(4)
        $This.Feature          = $This.New(5)
        $This.Application      = $This.New(6)
        $This.Event            = $This.New(7)
        $This.Task             = $This.New(8)
        $This.AppX             = $This.New(9)

        $This.Processor        = $This.New(10)
        $This.Processor.Refresh()
        $This.Disk              = $This.New(11)
        $This.Disk.Refresh()

        $This.Network           = $This.New(12)
        $This.Network.Refresh()

    }
    System([Object]$In)
    {
        $This.Mode              = 1

        $This.Snapshot          = $This.Load( 0,$In.Get(0))
        $This.BiosInformation   = $This.Load( 1,$In.Get(1))
        $This.ComputerSystem    = $This.Load( 2,$In.Get(2))
        $This.OperatingSystem   = $This.Load( 3,$In.Get(3))
        $This.HotFix            = $This.Load( 4,$In.Get(4))
        $This.Feature           = $This.Load( 5,$In.Get(5))
        $This.Application       = $This.Load( 6,$In.Get(6))
        $This.Event             = $This.Load( 7,$In.Get(7))
        $This.Task              = $This.Load( 8,$In.Get(8))
        $This.AppX              = $This.Load( 9,$In.Get(9))
        $This.Processor         = $This.Load(10,$In.Get(10))
        $This.Disk              = $This.Load(11,$In.Get(11))
        $This.Network           = $This.Load(12,$In.Get(12))
    }
    [Object] New([UInt32]$Rank)
    {
        $Item = Switch ($Rank)
        {
            00 {                   [Snapshot]::New() }
            01 {            [BiosInformation]::New() }
            02 {             [ComputerSystem]::New() }
            03 {            [OperatingSystem]::New() }
            04 {                [HotFixList]::New() }
            05 { [WindowsOptionalFeatureList]::New() }
            06 {            [ApplicationList]::New() }
            07 {       [EventLogProviderList]::New() }
            08 {          [ScheduledTaskList]::New() }
            09 {                  [AppXList]::New() }
            10 {             [ProcessorList]::New() }
            11 {                  [DiskList]::New() }
            12 {               [NetworkList]::New() }
        }

        Return $Item
    }
    [Object] Get([UInt32]$Index)
    {
        $Item = Switch ($Index)
        {
            00 { $This.Snapshot           }
            01 { $This.BiosInformation    }
            02 { $This.ComputerSystem     }
            03 { $This.OperatingSystem    }
            04 { $This.HotFix.Output      }
            05 { $This.Feature.Output     }
            06 { $This.Application.Output }
            07 { $This.Event.Output       }
            08 { $This.Task.Output        }
            09 { $This.AppX.Output        }
            10 { $This.Processor.Output   }
            11 { $This.Disk.Output        }
            12 { $This.Network.Output     }
            13 { $This.Event.Output       }
        }

        Return $Item
    }
    [Object] Load([UInt32]$Rank,[Object]$Object)
    {
        $Item = Switch ($Rank)
```

```
            {
                00 {                   [Snapshot]::New($Object) }
                01 {            [BiosInformation]::New($Object) }
                02 {             [ComputerSystem]::New($Object) }
                03 {            [OperatingSystem]::New($Object) }
                04 {                 [HotFixList]::New($Object) }
                05 { [WindowsOptionalFeatureList]::New($Object) }
                06 {            [ApplicationList]::New($Object) }
                07 {       [EventLogProviderList]::New($Object) }
                08 {          [ScheduledTaskList]::New($Object) }
                09 {                   [AppXList]::New($Object) }
                10 {              [ProcessorList]::New($Object) }
                11 {                   [DiskList]::New($Object) }
                12 {                [NetworkList]::New($Object) }
            }

            Return $Item
        }
        [Object] SystemProperty([UInt32]$Index,[UInt32]$Rank,[String]$Source,[String]$Name,[Object]$Value)
        {
            Return [SystemProperty]::New($Index,$Rank,$Source,$Name,$Value)
        }
        [Object] OutputSection([String]$Index,[String]$Name,[UInt32]$Slot)
        {
            Return [OutputSection]::New($Index,$Name,$Slot)
        }
        [Object] OutputFile()
        {
            $Out = @( )
            ForEach ($Name in $This.PSObject.Properties.Name)
            {
                $Slot = Switch -Regex ($Name) { "(^Snap|^Bios|^Comp|^Oper)" { 0 } Default { 1 } "^Disk" { 2 } }
                $Section = $This.OutputSection($Out.Count,$Name,$Slot)

                Switch ($Name)
                {
                    Snapshot
                    {
                        ForEach ($Item in $This.Snapshot[0].PSObject.Properties)
                        {
                            $Section.Add($Item.Name,$Item.Value)
                        }
                    }
                    BiosInformation
                    {
                        ForEach ($Item in $This.BiosInformation[0].PSObject.Properties)
                        {
                            $Section.Add($Item.Name,$Item.Value)
                        }
                    }
                    ComputerSystem
                    {
                        ForEach ($Item in $This.ComputerSystem[0].PSObject.Properties)
                        {
                            $Section.Add($Item.Name,$Item.Value)
                        }
                    }
                    OperatingSystem
                    {
                        ForEach ($Item in $This.OperatingSystem[0].PSObject.Properties)
                        {
                            $Section.Add($Item.Name,$Item.Value)
                        }
                    }
                    HotFix
                    {
                        ForEach ($Item in $This.HotFix.Output)
                        {
                            $Section.Add($Item.Tag(),$Item.Value())
                        }
                    }
                    Feature
                    {
                        ForEach ($Item in $This.Feature.Output)
                        {
                            $Section.Add($Item.Tag(),$Item.Value())
                        }
                    }
                    Application
                    {
                        ForEach ($Item in $This.Application.Output)
                        {
                            $Section.Add($Item.Tag(),$Item.Value())
                        }
                    }
                    Event
                    {
                        ForEach ($Item in $This.Event.Output)
                        {
                            $Section.Add($Item.Tag(),$Item.Value())
                        }
                    }
                    Task
                    {
                        ForEach ($Item in $This.Task.Output)
                        {
                            $Section.Add($Item.Tag(),$Item.Value())
                        }
                    }
                    AppX
                    {
                        ForEach ($Item in $This.AppX.Output)
                        {
                            $Section.Add($Item.Tag(),$Item.Value())
                        }
                    }
                    Processor
                    {
                        $Rank = 0
                        ForEach ($Processor in $This.Processor.Output)
                        {
                            $Section.Add("Processor$Rank",":")
                            ForEach ($Item in $Processor.PSObject.Properties)
                            {
                                $Section.Add($Item.Name,$Item.Value)
                            }
                            $Rank ++
                        }
                    }
                    Disk
                    {
                        $Rank = 0
                        ForEach ($Disk in $This.Disk.Output)
                        {
                            $Section.Add("Disk$Rank",":")
                            ForEach ($Item in $Disk.PSObject.Properties)
                            {
                                Switch ($Item.Name)
                                {
                                    Partition
                                    {
                                        $Part = 0
                                        ForEach ($Partition in $Disk.Partition.Output)
                                        {
                                            $Section.Add("Disk$Rank.Partition$Part",":")
                                            ForEach ($Prop in $Partition.PSObject.Properties)
                                            {
                                                $Section.Add($Prop.Name,$Prop.Value)
                                            }
                                            $Part ++
                                        }
                                    }
                                    Volume
                                    {
```

```
                                $Vol = 0
                                ForEach ($Volume in $Disk.Volume.Output)
                                {
                                    $Section.Add("Disk$Rank.Volume$Vol",":")
                                    ForEach ($Prop in $Volume.PSObject.Properties)
                                    {
                                        $Section.Add($Prop.Name,$Prop.Value)
                                    }
                                    $Vol ++
                                }
                            }
                            Default
                            {
                                $Section.Add($Item.Name,$Item.Value)
                            }
                        }
                    }
                }
            }
            Network
            {
                $Rank = 0
                ForEach ($Network in $This.Network.Output)
                {
                    $Section.Add("Network$Rank",":")
                    ForEach ($Item in $Network.PSObject.Properties)
                    {
                        $Section.Add($Item.Name,$Item.Value)
                    }
                    $Rank ++
                }
            }
        }
        $Out += $Section
    }

    Return [OutputFile]::New($Out).Output
}
WriteOutput()
{
    If ($This.Mode -ne 0)
    {
        Throw "Invalid mode"
    }

    $Target = "{0}\{1}.txt" -f [Environment]::GetEnvironmentVariable("Temp"), $This.Snapshot.DisplayName
    $This.WriteOutput($Target)
}
WriteOutput([String]$Target)
{
    $Parent = $Target | Split-Path

    If ($This.Mode -ne 0)
    {
        Throw "Invalid mode"
    }

    ElseIf (![System.IO.Directory]::Exists($Parent))
    {
        Throw "Invalid path"
    }

    ElseIf ([System.IO.File]::Exists($Target))
    {
        Throw "File already exists"
    }

    $Value = $This.OutputFile()

    Try
    {
        [System.IO.File]::WriteAllLines($Target,$Value)
```

```
                If (![System.IO.File]::Exists($Target))
                {
                    Throw "Exception [!] File was not saved [!]"
                }
                Else
                {
                    [Console]::WriteLine("File [$Target] saved.")
                }
            }
            Catch
            {
                Write-Error "An unknown error occurred."
            }
        }
        [String] ToString()
        {
            Return "{0}, {1} | {2}, {3} {4}-{5}" -f $This.Snapshot.ComputerName,
            $This.ComputerSystem.Manufacturer,
            $This.ComputerSystem.Model,
            $This.OperatingSystem.Caption,
            $This.OperatingSystem.Version,
            $This.OperatingSystem.Build
        }
    }
```

```
                                                                                                          ______________/
\________________________________________________________________________________________________________/ Class [System]
  Output /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯
```

In this section, I'll cover some of the output from the function using information from a virtual machine. Running the function is as easy as running

```
$System = Get-FESystem -Mode 0
$System = Get-FESystem -Mode 1 -Path $Path
$System = Get-FESystem -Mode 2 -InputObject (Get-Content $Path)
```

Though the mode is currently disabled, it is specifically meant to allow the function to switch between logging and reporting to the console, or for silent operation.

The trick is being able to get a lot of information rather instantly, and there aren't a lot of ways to do that without having files already written to the disk that the operating system uses, in order to get that info.

Windows has various information written to disk, which is why things like [msinfo32.exe], or [services.msc] are pretty quick about the information they collect.

The first option will collect all of the information in the class, seen below.

```
PS Prompt:\> $System

Snapshot          : SERVER01
BiosInformation   : Microsoft Corporation | Hyper-V UEFI Release v4.0
ComputerSystem    : Microsoft Corporation | Virtual Machine
OperatingSystem   : Microsoft Windows Server 2016 Datacenter Evaluation 10.0.14393-14393
HotFix            : (6) <FESystem.HotFixList>
Feature           : (322) <FESystem.WindowsOptionalFeatureList>
Application       : (1) <FESystem.ApplicationList>
Event             : (375) <FESystem.EventLogProviderList>
Task              : (126) <FESystem.ScheduledTaskList>
AppX              : (0) <FESystem.AppXList>
Processor         : (1) <FESystem.ProcessorList>
Disk              : (1) <FESystem.DiskList>
Network           : (14) <FESystem.NetworkList>

PS Prompt:\>
```

This information probably looks very similar to the output of Get-FENetwork (which I covered recently), and that is because I'm using a methodology to create the console objects AND, format them cleanly and eleganty.

While ALL of these properties are their own object, some of them are different.

The first (4) items are objects with their own methods and properties, they're not list type items.
There's no (number) next to them, but rather, a simplified version of the information within those classes.

If we were to expand the first (4) properties, we would get this.

```
PS Prompt:\> $System.Snapshot

Start        : 12/30/2022 10:06:45
ComputerName : SERVER01
Name         : server01
DisplayName  : 2022-1230-100645-SERVER01
DNS          : -
NetBIOS      : server01
Hostname     : server01
Username     : Administrator
Principal    : System.Security.Principal.WindowsPrincipal
IsAdmin      : True
Caption      : Microsoft Windows Server 2016 Datacenter Evaluation
Version      : 2022.12.1
ReleaseID    : 1607
Build        : 14393
Code         : Redstone 1
Description  : Anniversary Update
SKU          : -
Chassis      : Mobile/Laptop
Guid         : 40415178-5404-4354-9b3d-025ba9fcf313
Complete     : 0
Elapsed      :

PS Prompt:\>
PS Prompt:\> $System.BiosInformation

Name            : Hyper-V UEFI Release v4.0
Manufacturer    : Microsoft Corporation
SerialNumber    : 5150-5855-7329-9070-4952-1001-40
Version         : VRTUAL - 1
ReleaseDate     : 10/31/2019 20:00:00
SmBiosPresent   : True
SmBiosVersion   : Hyper-V UEFI Release v4.0
SmBiosMajor     : 3
SmBiosMinor     : 1
SystemBiosMajor : 4
SystemBiosMinor : 0

PS Prompt:\>
PS Prompt:\> $System.ComputerSystem

Manufacturer : Microsoft Corporation
Model        : Virtual Machine
Product      : Hyper-V UEFI Release v4.0
Serial       : 5150-5855-7329-9070-4952-1001-40
Memory       : 2.70 GB
Architecture : x64
UUID         : EC298D66-FDFE-4205-8285-842DB8431DE0
Chassis      : Desktop
BiosUefi     : UEFI
AssetTag     : 5150-5855-7329-9070-4952-1001-40

PS Prompt:\>
PS Prompt:\> $System.OperatingSystem

Caption  : Microsoft Windows Server 2016 Datacenter Evaluation
Version  : 10.0.14393
Build    : 14393
Serial   : 00377-10000-00000-AA360
Language : 1033
```

```
Product  : 400
Type     : 18

PS Prompt:\>
```

The rest of the items are all list type items, and they each have an outer container, and then internal objects that can range from <none>, <single>, or <multiple> object(s).

```
PS Prompt:\> $System.HotFix

Name   Count Output
----   ----- ------
HotFix     6 {<FESystem.HotFixItem>, <FESystem.HotFixItem>, <FESystem.HotFixItem>, <FESystem.HotFixItem>...}

PS Prompt:\>
```

```
PS Prompt:\> $System.HotFix.Output

HotFixID  Description     InstalledBy         InstalledOn
--------  -----------     -----------         -----------
KB3192137 Update                              09/12/2016
KB3211320 Update                              01/07/2017
KB4589210 Update          NT AUTHORITY\SYSTEM 12/17/2022
KB5012170 Security Update NT AUTHORITY\SYSTEM 12/16/2022
KB5017396 Security Update NT AUTHORITY\SYSTEM 12/15/2022
KB5021235 Security Update NT AUTHORITY\SYSTEM 12/18/2022

PS Prompt:\>
```

```
PS Prompt:\> $System.Feature

Name              Count Output
----              ----- ------
Optional Features   322 {<FESystem.WindowsOptionalFeatureItem>, <FESystem.WindowsOptionalFeatureItem>...}

PS Prompt:\>
```

```
PS Prompt:\> $System.Feature.Output[0]

FeatureName                State
-----------                -----
ActiveDirectory-PowerShell Disabled

PS Prompt:\>
```

```
PS Prompt:\> $System.Application

Name        Count Output
----        ----- ------
Application     1 {<FESystem.ApplicationItem>}

PS Prompt:\>
```

```
PS Prompt:\> $System.Application.Output

Type DisplayName                  DisplayVersion
---- -----------                  --------------
MSI  Microsoft Visual Studio Code 1.74.2

PS Prompt:\>
```

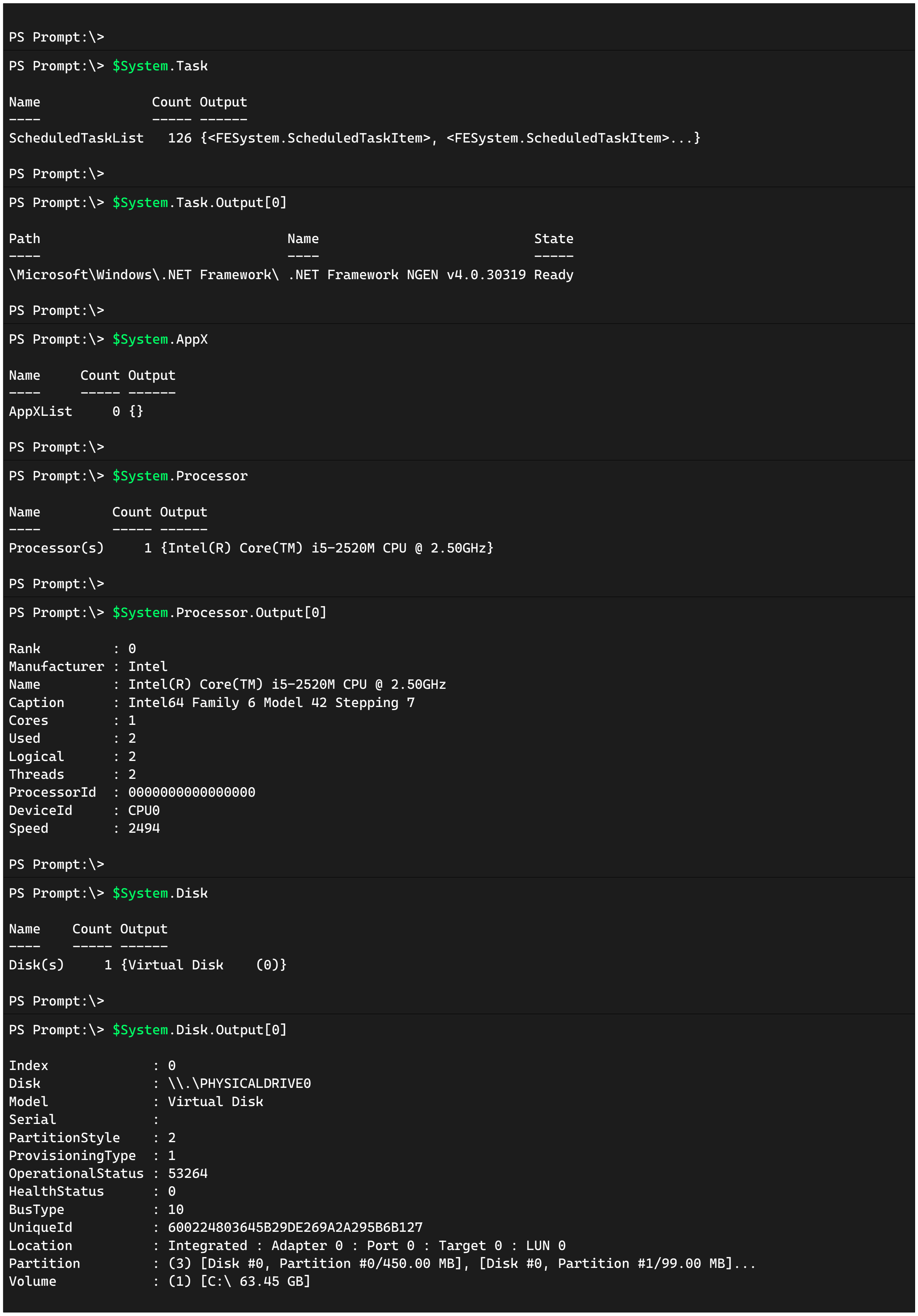
```
PS Prompt:\> $System.Event

Name       Count Output
----       ----- ------
Event Logs   375 {<FESystem.EventLogProviderItem>, <FESystem.EventLogProviderItem>...}

PS Prompt:\>
```

```
PS Prompt:\> $System.Event.Output[0]

Name      DisplayName
----      -----------
Provider0 Application
```



```
PS Prompt:\>
PS Prompt:\> $System.Task

Name              Count Output
----              ----- ------
ScheduledTaskList   126 {<FESystem.ScheduledTaskItem>, <FESystem.ScheduledTaskItem>...}

PS Prompt:\>
PS Prompt:\> $System.Task.Output[0]

Path                               Name                         State
----                               ----                         -----
\Microsoft\Windows\.NET Framework\ .NET Framework NGEN v4.0.30319 Ready

PS Prompt:\>
PS Prompt:\> $System.AppX

Name     Count Output
----     ----- ------
AppXList     0 {}

PS Prompt:\>
PS Prompt:\> $System.Processor

Name         Count Output
----         ----- ------
Processor(s)     1 {Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz}

PS Prompt:\>
PS Prompt:\> $System.Processor.Output[0]

Rank         : 0
Manufacturer : Intel
Name         : Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz
Caption      : Intel64 Family 6 Model 42 Stepping 7
Cores        : 1
Used         : 2
Logical      : 2
Threads      : 2
ProcessorId  : 0000000000000000
DeviceId     : CPU0
Speed        : 2494

PS Prompt:\>
PS Prompt:\> $System.Disk

Name    Count Output
----    ----- ------
Disk(s)     1 {Virtual Disk    (0)}

PS Prompt:\>
PS Prompt:\> $System.Disk.Output[0]

Index             : 0
Disk              : \\.\PHYSICALDRIVE0
Model             : Virtual Disk
Serial            :
PartitionStyle    : 2
ProvisioningType  : 1
OperationalStatus : 53264
HealthStatus      : 0
BusType           : 10
UniqueId          : 600224803645B29DE269A2A295B6B127
Location          : Integrated : Adapter 0 : Port 0 : Target 0 : LUN 0
Partition         : (3) [Disk #0, Partition #0/450.00 MB], [Disk #0, Partition #1/99.00 MB]...
Volume            : (1) [C:\ 63.45 GB]
```

```
PS Prompt:\>
PS Prompt:\> $System.Disk.Output[0].Partition

Name         Count Output
----         ----- ------
Partition(s)     3 {[Disk #0, Partition #0/450.00 MB], [Disk #0, Partition #1/99.00 MB]l...}

PS Prompt:\>
PS Prompt:\> $System.Disk.Output[0].Partition.Output[0]

Rank      : 0
Type      : GPT: Unknown
Name      : Disk #0, Partition #0
Size      : 450.00 MB
Boot      : 0
Primary   : 0
Disk      : 0
Partition : 0

PS Prompt:\>
PS Prompt:\> $System.Disk.Output[0].Volume

Name    Count Output
----    ----- ------
Volumes     1 {[C:\ 63.45 GB]}

PS Prompt:\>
PS Prompt:\> $System.Disk.Output[0].Volume.Output[0]

Rank         : 0
DriveID      : C:
Description  : Local Fixed Disk
Filesystem   : NTFS
Partition    : Disk #0, Partition #2
VolumeName   :
VolumeSerial : 64B8FC7B
Size         : 63.45 GB
Freespace    : 50.20 GB
Used         : 13.25 GB

PS Prompt:\>
PS Prompt:\> $System.Network

Name       Count Output
----       ----- ------
Network(s)    14 {Microsoft Hyper-V Network Adapter, Microsoft Kernel Debug Network Adapter,...}

PS Prompt:\>
PS Prompt:\> $System.Network.Output[0]

Rank       : 0
Name       : Microsoft Hyper-V Network Adapter
Status     : 1
IPAddress  : 172.27.241.94
SubnetMask : 255.255.240.0
Gateway    : 172.27.240.1
DnsServer  : 172.27.240.1
DhcpServer : 172.27.240.1
MacAddress : 00:15:5D:F4:7F:00

PS Prompt:\>
```

The last thing I'll talk about in this document before I conclude, is that this variable `$System` has a lot of functionality underneath all of this information. Simply put, there is definitely plenty more that can be done with it... and some of the classes handle serialization and deserialization.

`$Output = $System.OutputFile()`
That right there will cast all of the information on a given system to a format that can be saved as a file.

That method does not actually SAVE the file to the drive, however. All it does, is cleanly organize all of the information necessary to generate an output file that can be used as an input file elsewhere.

```
===============================================================================================================
[Snapshot]
===============================================================================================================

Start            : 12/30/2022 10:06:45
ComputerName     : SERVER01
Name             : server01
DisplayName      : 2022-1230-100645-SERVER01
DNS              : -
NetBIOS          : server01
Hostname         : server01
Username         : Administrator
Principal        : System.Security.Principal.WindowsPrincipal
IsAdmin          : True
Caption          : Microsoft Windows Server 2016 Datacenter Evaluation
Version          : 2022.12.1
ReleaseID        : 1607
Build            : 14393
Code             : Redstone 1
Description      : Anniversary Update
SKU              : -
Chassis          : Mobile/Laptop
Guid             : 40415178-5404-4354-9b3d-025ba9fcf313
Complete         : 0
Elapsed          :

===============================================================================================================
[BiosInformation]
===============================================================================================================

Name             : Hyper-V UEFI Release v4.0
Manufacturer     : Microsoft Corporation
SerialNumber     : 5150-5855-7329-9070-4952-1001-40
Version          : VRTUAL - 1
ReleaseDate      : 10/31/2019 20:00:00
SmBiosPresent    : True
SmBiosVersion    : Hyper-V UEFI Release v4.0
SmBiosMajor      : 3
SmBiosMinor      : 1
SystemBiosMajor  : 4
```

Those are just the first (40) lines or so, but it does go into the nitty gritty of having to figure out all of the property lengths, where to insert labels in order to reconsitute the objects on another system, etc.

```
                                                                                                      ________/
\____________________________________________________________________________________________________/ Output
  Conclusion /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯
```

At any rate, this function is just a small portion of the things needed to run the DCPromo function that I have been slowly waiting to finish and update, and then... New-FEInfrastructure is bound to get some more attention.
Then, I may update the module officially.
Every single time that I think to start updating components of the module...?
It takes me quite a long time to do... because as I touch up the things I've worked on before, I am ALWAYS looking for more efficient design implementations, or something that allows the process to be more seamless, flexible, organized, or capable...
And that is the part that I put a lot of time and thought into, because anyone who has a lot of pride in their work is going to be extremely attentive to detail and looking for ways to solve problems in a parallel manner.

```
                                                                                                  ____________/
\________________________________________________________________________________________________/ Conclusion

  ________________________________________________________
 |¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯|
 |                                    Michael C. Cook Sr. |
 |                                      Security Engineer |
 |                                   Secure Digits Plus LLC |
 |________________________________________________________|
  ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```