```
 ____    ____________________________________________________________________________________________________
//¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
\\__//¯¯¯ New-VmController [~] 05/03/2023                                                          ___//¯¯\\
 ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
     ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
```

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

    In this particular document, I'm going to cover a function that I have been
    developing to manage virtual machines using [Hyper-V] + [PowerShell], that has a
    graphical user interface that uses [XAML/Extensible Application Markup Language].

    First, I'm going to cover the actual function as it currently is, at
    [2023-05-02 20:13:58] with hash value:
    [70F3DB5EE74D52DBA590D23C2EC5320316AB8EBC26F07030BE77ADA79F4137CC]

    What I will do in the first block is put the function wrapper.
    Then, I will cover each individual class.
    Then, I will cover the output from the class.
    Then, I may also cover the script that is featured in the video...
    <insert video link #1>
    <insert video link #2>

```
                                                                                                          ______________/
\________________________________________________________________________________________________________/ Introduction
  Function /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯
```

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

 //==================================================================================================\\
//  Module     : [FightingEntropy()][2023.4.0]                                                         \\
\\  Date       : 2023-05-02 20:13:58                                                                   //
 \\==================================================================================================//

    FileName   : New-VmController.ps1
    Solution   : [FightingEntropy()][2023.4.0]
    Purpose    : Creates a [PowerShell] object that can optionally initialize a
                 (GUI/graphical user interface) to orchestrate the networking, credentials,
                 imaging, templatization, deployment and configuration of (a single/multiple)
                 [virtual machines] in [Hyper-V].
    Author     : Michael C. Cook Sr.
    Contact    : @mcc85s
    Primary    : @mcc85s
    Created    : 2023-04-29
    Modified   : 2023-05-02
    Demo       : N/A
    Version    : 0.0.0 - () - Finalized functional version 1
    TODO       : N/A

.Example
#>

Function New-VmController
{
    <# [Insert classes here] #>

    [VmControllerMaster]::New()
}
```

```
                                                                                                              __________/
\____________________________________________________________________________________________________________/ Function
  Class [ImageLabel] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

This object is used to [categorize] the [queue objects] whenever the utility is using it to extract (*.wim) files from multiple images. It is not used in this specific (*video/document/function*), but it is from the [ImageController] in [FightingEntropy(π)][FEInfrastructure] video [https://youtu.be/6yQr06_rA4I]

```
Class ImageLabel
{
    [UInt32]           $Index
    [String]            $Name
    [String]            $Type
    [String]         $Version
    [UInt32[]] $SelectedIndex
    [Object[]]       $Content
    ImageLabel([UInt32]$Index,[Object]$Selected,[UInt32[]]$Queue)
    {
        $This.Index         = $Index
        $This.Name          = $Selected.Fullname
        $This.Type          = $Selected.Type
        $This.Version       = $Selected.Version
        $This.SelectedIndex = $Queue
        $This.Content       = @($Selected.Content | ? Index -in $Index)
        ForEach ($Item in $This.Content)
        {
            $Item.Type      = $Selected.Type
            $Item.Version   = $Selected.Version
        }
    }
    [String] ToString()
    {
        Return "<FEModule.Image[Label]>"
    }
}
```

```
                                                                                                    ____________________/
\___________________________________________________________________________________________________/ Class [ImageLabel]
  Class [ImageByteSize] /-------------------------------------------------------------------------------------------------\
/-----------------------
```

This class is essentially meant to make a byte size [UInt64] more consumable to look at.

```
Class ImageByteSize
{
    [String]   $Name
    [UInt64]  $Bytes
    [String]   $Unit
    [String]   $Size
    ImageByteSize([String]$Name,[UInt64]$Bytes)
    {
        $This.Name   = $Name
        $This.Bytes  = $Bytes
        $This.GetUnit()
        $This.GetSize()
    }
    GetUnit()
    {
        $This.Unit   = Switch ($This.Bytes)
        {
            {$_ -lt 1KB}                 {     "Byte" }
            {$_ -ge 1KB -and $_ -lt 1MB} { "Kilobyte" }
            {$_ -ge 1MB -and $_ -lt 1GB} { "Megabyte" }
            {$_ -ge 1GB -and $_ -lt 1TB} { "Gigabyte" }
            {$_ -ge 1TB}                 { "Terabyte" }
        }
    }
    GetSize()
    {
        $This.Size   = Switch -Regex ($This.Unit)
```

```
            {
                ^Byte     {     "{0} B" -f  $This.Bytes/1    }
                ^Kilobyte { "{0:n2} KB" -f ($This.Bytes/1KB) }
                ^Megabyte { "{0:n2} MB" -f ($This.Bytes/1MB) }
                ^Gigabyte { "{0:n2} GB" -f ($This.Bytes/1GB) }
                ^Terabyte { "{0:n2} TB" -f ($This.Bytes/1TB) }
            }
        }
        [String] ToString()
        {
            Return $This.Size
        }
    }
```

```
PS Prompt:\> [ImageByteSize]::New("Example",8675309)

Name      Bytes Unit     Size
----      ----- ----     ----
Example 8675309 Megabyte 8.27 MB

PS Prompt:\>
```

```
                                                                                                ______________________/
\______________________________________________________________________________________________/ Class [ImageByteSize]
  Class [ImageEdition] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

This is the object extracted from the [WindowsImage] data return objects.

```
    Class ImageEdition
    {
        Hidden [Object] $ImageFile
        Hidden [Object]      $Arch
        [UInt32]            $Index
        [String]             $Type
        [String]          $Version
        [String]             $Name
        [String]      $Description
        [Object]             $Size
        [UInt32]     $Architecture
        [String]  $DestinationName
        [String]            $Label
        ImageEdition([Object]$Path,[Object]$Image,[Object]$Slot)
        {
            $This.ImageFile    = $Path
            $This.Arch         = $Image.Architecture
            $This.Type         = $Image.InstallationType
            $This.Version      = $Image.Version
            $This.Index        = $Slot.ImageIndex
            $This.Name         = $Slot.ImageName
            $This.Description  = $Slot.ImageDescription
            $This.Size         = $This.SizeBytes($Slot.ImageSize)
            $This.Architecture = @(86,64)[$This.Arch -eq 9]

            $This.GetLabel()
        }
        [Object] SizeBytes([UInt64]$Bytes)
        {
            Return [ImageByteSize]::New("Image",$Bytes)
        }
        GetLabel()
        {
            $Number = $Null
            $Tag    = $Null
            Switch -Regex ($This.Name)
            {
                Server
                {
                    $Number               = [Regex]::Matches($This.Name,"(\d{4})").Value
                    $Edition              = [Regex]::Matches($This.Name,"(Standard|Datacenter)").Value
                    $Tag                  = @{ Standard = "SD"; Datacenter = "DC" }[$Edition]

                    If ($This.Name -notmatch "Desktop")
                    {
                        $Tag += "X"
                    }

                    $This.DestinationName = "Windows Server $Number $Edition (x64)"
                }
                Default
                {
                    $Number               = [Regex]::Matches($This.Name,"(\d+)").Value
                    $Edition              = $This.Name -Replace "Windows \d+ ",''
                    $Tag                  = Switch -Regex ($Edition)
                    {
                        "^Home$"             { "HOME"       } "^Home N$"            { "HOME_N"   }
                        "^Home Sin.+$"       { "HOME_SL"    } "^Education$"         { "EDUC"     }
                        "^Education N$"      { "EDUC_N"     } "^Pro$"               { "PRO"      }
                        "^Pro N$"            { "PRO_N"      } "^Pro Education$"     { "PRO_EDUC" }
                        "^Pro Education N$"  { "PRO_EDUC_N" } "^Pro for Work.+$"    { "PRO_WS"   }
                        "^Pro N for Work.+$" { "PRO_N_WS"   } "Enterprise"          { "ENT"      }
                    }

                    $This.DestinationName = "{0} (x{1})" -f $This.Name, $This.Architecture
                }
            }

            $This.Label           = "{0}{1}{2}-{3}" -f $Number, $Tag, $This.Architecture, $This.Version
        }
        [String] ToString()
        {
            Return "<FEModule.Image[Edition]>"
        }
    }
```

```
PS Prompt:\> $Vm.Image.Edition

Index           : 6
Type            : Client
Version         : 10.0.22621.525
Name            : Windows 11 Pro
Description     : Windows 11 Pro
Size            : 15.35 GB
Architecture    : 64
DestinationName : Windows 11 Pro (x64)
Label           : 11PRO64-10.0.22621.525

PS Prompt:\>
```

```
                                                                                                 ______________________/
\_______________________________________________________________________________________________/ Class [ImageEdition]
  Class [ImageFile] /-------------------------------------------------------------------------------------------------\
/-------------------
```

```
PS Prompt:\> $Vm.Image.File

Index    : 0
Type     : Windows
Version  : 10.0.22621.525
Name     : Win11_22H2_English_x64v1.iso
Fullname : C:\Images\Win11_22H2_English_x64v1.iso

PS Prompt:\>
```

```
Class ImageFile
{
    [UInt32]             $Index
    [String]              $Type
    [String]           $Version
    [String]              $Name
    [String]          $Fullname
    Hidden [String]     $Letter
    Hidden [Object[]]  $Content
    ImageFile([UInt32]$Index,[String]$Fullname)
    {
        $This.Index     = $Index
        $This.Name      = $Fullname | Split-Path -Leaf
        $This.Fullname  = $Fullname
        $This.Content   = @( )
    }
    [Object] GetDiskImage()
    {
        Return Get-DiskImage -ImagePath $This.Fullname
    }
    [String] DriveLetter()
    {
        Return $This.GetDiskImage() | Get-Volume | % DriveLetter
    }
    MountDiskImage()
    {
        If ($This.GetDiskImage() | ? Attached -eq 0)
        {
            Mount-DiskImage -ImagePath $This.Fullname
        }

        Do
        {
            Start-Sleep -Milliseconds 100
        }
        Until ($This.GetDiskImage() | ? Attached -eq 1)

        $This.Letter = $This.DriveLetter()
    }
    DismountDiskImage()
    {
        Dismount-DiskImage -ImagePath $This.Fullname
    }
    [Object[]] InstallWim()
    {
        Return ("{0}:\" -f $This.Letter | Get-ChildItem -Recurse | ? Name -match "^install\.(wim|esd)")
    }
    [String] ToString()
    {
        Return "<FEModule.Image[File]>"
    }
}
```

```
                                                                                                    __________________/
\__________________________________________________________________________________________________/ Class [ImageFile]
  Class [ImageObject] /-----------------------------------------------------------------------------------------------\
/---------------------
```

Strictly meant for [serialization + deserialization] via the [template file].

```
Class ImageObject
{
    [Object] $File
    [Object] $Edition
    ImageObject([Object]$File)
    {
        $This.File    = $File
        $This.Edition = $Null
    }
    ImageObject([Object]$File,[Object]$Edition)
    {
        $This.File    = $File
        $This.Edition = $Edition
    }
    [String] ToString()
    {
        Return $This.File.Fullname
    }
}
```

```
PS Prompt:\> $Vm.Image

File                         Edition
----                         -------
<FEVirtual.VmNodeImage[File] <FEVirtual.VmNodeImage[Edition]

PS Prompt:\>
```

```
                                                                                                  _____________________/
\________________________________________________________________________________________________/ Class [ImageObject]
  Class [ImageController] /--------------------------------------------------------------------------------------------\
/-------------------------
```

The main [ImageController] class, is meant to import a series of (*.iso) files, whether they are [Linux], [FreeBSD], [MacOS (*eventually*)] or [Windows]. When they are [Windows] images, it will open them and determine what [ImageEdition]'s are in the [ImageFile].

```
    Class ImageController
    {
        [String]        $Source
        [String]        $Target
        [Int32]       $Selected
        [Object]         $Store
        [Object]         $Queue
        [Object]          $Swap
        [Object]        $Output
        Hidden [String] $Status
        ImageController()
        {
            $This.Source   = $Null
            $This.Target   = $Null
            $This.Selected = $Null
            $This.Store    = @( )
            $This.Queue    = @( )
        }
        Clear()
        {
            $This.Selected = -1
            $This.Store    = @( )
            $This.Queue    = @( )
        }
        [Object] ImageLabel([UInt32]$Index,[Object]$Selected,[UInt32[]]$Queue)
        {
            Return [ImageLabel]::New($Index,$Selected,$Queue)
        }
        [Object] ImageEdition([Object]$Fullname,[Object]$Image,[Object]$Slot)
        {
            Return [ImageEdition]::New($Fullname,$Image,$Slot)
        }
        [Object] ImageFile([UInt32]$Index,[String]$Fullname)
        {
            Return [ImageFile]::New($Index,$Fullname)
        }
        [Object] ImageObject([Object]$Image)
        {
            Return [ImageObject]::New($Image)
        }
        [Object] ImageObject([Object]$Image,[Object]$Edition)
        {
            Return [ImageObject]::New($Image,$Edition)
        }
        [Object[]] GetContent()
        {
            If (!$This.Source)
            {
                Throw "Source path not set"
            }

            Return Get-ChildItem -Path $This.Source *.iso
        }
        GetWindowsImage([String]$Path)
        {
            $File         = $This.Current()
            $Image        = Get-WindowsImage -ImagePath $Path -Index 1
            $File.Version = $Image.Version

            $File.Content = ForEach ($Item in Get-WindowsImage -ImagePath $Path)
            {
                $This.ImageEdition($Path,$Image,$Item)
            }
        }
        Select([UInt32]$Index)
        {
            If ($Index -gt $This.Store.Count)
            {
                Throw "Invalid index"
            }

            $This.Selected = $Index
        }
        SetSource([String]$Source)
        {
            If (![System.IO.Directory]::Exists($Source))
            {
                Throw "Invalid source path"
            }

            $This.Source = $Source
        }
        SetTarget([String]$Target)
        {
            If (![System.IO.Directory]::Exists($Target))
            {
                $Parent = Split-Path $Target -Parent
                If (![System.IO.Directory]::Exists($Parent))
                {
                    Throw "Invalid target path"
                }

                [System.IO.Directory]::CreateDirectory($Target)
            }

            $This.Target = $Target
        }
        Refresh()
        {
            $This.Clear()

            ForEach ($Item in $This.GetContent())
            {
                $This.Add($Item.Fullname)
            }
        }
        Add([String]$File)
        {
            $This.Store += $This.ImageFile($This.Store.Count,$File)
        }
        [Object] Current()
        {
            If ($This.Selected -eq -1)
            {
                Throw "No image selected"
            }

            Return $This.Store[$This.Selected]
        }
        Load()
        {
            If (!$This.Current().GetDiskImage().Attached)
            {
                $This.Current().MountDiskImage()
            }
        }
        Unload()
        {
            If (!!$This.Current().GetDiskImage().Attached)
            {
                $This.Current().DismountDiskImage()
            }
        }
        ProcessSlot()
        {
            $Current         = $This.Current()
            $This.Status     = "Loading [~] {0}" -f $Current.Name
            $This.Load()

            $File            = $Current.InstallWim()
            $Current.Type    = @("Non-Windows","Windows")[$File.Count -ne 0]
            $This.Status     = "Type [+] {0}" -f $Current.Type

            If ($Current.Type -eq "Windows")
            {
                If ($File.Count -gt 1)
                {
                    $File        = $File | ? Fullname -match x64
                }

                $This.GetWindowsImage($File.Fullname)
            }

            $This.Status     = "Unloading [~] {0}" -f $Current.Name
            $This.Unload()
        }
        Chart()
        {
            Switch ($This.Store.Count)
            {
                0
                {
                    Throw "No images detected"
                }
                1
                {
                    $This.Select(0)
                    $This.ProcessSlot()
                }
                Default
                {
                    ForEach ($X in 0..($This.Store.Count-1))
                    {
                        $This.Select($X)
                        $This.ProcessSlot()
                    }
                }
            }
        }
        AddQueue([UInt32[]]$Queue)
        {
            If ($This.Current().Fullname -in $This.Queue.Name)
            {
```

```
            Throw "Image already in the queue, remove, and reindex"
        }

        $This.Queue += $This.ImageLabel($This.Queue.Count,$This.Current(),$Queue)
    }
    RemoveQueue([String]$Name)
    {
        If ($Name -in $This.Queue.Name)
        {
            $This.Queue = @($This.Queue | ? Name -ne $Name)
        }
    }
    Extract()
    {
        If (!$This.Target)
        {
            Throw "Must set target path"
        }

        ElseIf ($This.Queue.Count -eq 0)
        {
            Throw "No items queued"
        }

        $X = 0
        ForEach ($Queue in $This.Queue)
        {
            $Disc        = $This.Store | ? FullName -eq $Queue.Name
            If (!$Disc.GetDiskImage().Attached)
            {
                $This.Status = "Mounting [~] {0}" -f $Disc.Name
                $Disc.MountDiskImage()
                $Disc.Letter = $Disc.DriveLetter()
            }

            $Path         = $Disc.InstallWim()
            If ($Path.Count -gt 1)
            {
                $Path     = $Path | ? Name -match x64
            }

            ForEach ($File in $Disc.Content)
            {
                $ISO                        = @{

                    SourceIndex             = $File.Index
                    SourceImagePath         = $Path.Fullname
                    DestinationImagePath    = "{0}\({1}){2}\{2}.wim" -f $This.Target, $X, $File.Label
                    DestinationName         = $File.DestinationName
                }

                $Folder                     = $Iso.DestinationImagePath | Split-Path -Parent
                # Check + create folder
                If (![System.IO.Directory]::Exists($Folder))
                {
                    [System.IO.Directory]::CreateDirectory($Folder)
                }

                # Check + remove file
                If ([System.IO.File]::Exists($Iso.DestinationImagePath))
                {
                    [System.IO.File]::Delete($Iso.DestinationImagePath)
                }

                # Create the file
                $This.Status = "Extracting [~] $($File.DestinationName)"

                Export-WindowsImage @ISO | Out-Null
                $This.Status = "Extracted [~] $($This.DestinationName)"

                $X ++
            }
            $This.Status = "Dismounting [~] {0}" -f $Disc.Name
            $Disc.DismountDiskImage()
        }

        $This.Status = "Complete [+] ($($This.Queue.SelectedIndex.Count)) *.wim files Extracted"
    }
    [String] ToString()
    {
        Return "<FEModule.Image[Controller]>"
    }
}
```

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

Source   : C:\Images
Target   :
Selected : 0
Store    : {<FEModule.Image[File]>, <FEModule.Image[File]>}
Queue    : {}
Swap     :
Output   :

PS Prompt:\>
```

```
                                                                                                            ________________________/
\__________________________________________________________________________________________________________/ Class [ImageController]
  Enum [SecurityOptionType] /----------------------------------------------------------------------------------------------------------\
/---------------------------
```

    Meant for [Windows 10] installation, the available [security question types] are shortened to these monikers.

```
Enum SecurityOptionType
{
    FirstPet
    BirthCity
    ChildhoodNick
    ParentCity
    CousinFirst
    FirstSchool
}
```

```
                                                                                                          __________________________/
\________________________________________________________________________________________________________/ Enum [SecurityOptionType]
  Class [SecurityOptionItem] /---------------------------------------------------------------------------------------------------------\
/----------------------------
```

    This item hosts the information needed to select an option from the menu during the [Windows 10] installation.

```
Class SecurityOptionItem
{
    [UInt32]       $Index
    [String]        $Name
    [String] $Description
    SecurityOptionItem([String]$Name)
    {
        $This.Index = [UInt32][SecurityOptionType]::$Name
        $This.Name  = [SecurityOptionType]::$Name
    }
}
```

```
PS Prompt:\> $Security.Slot[0] | FL

Index       : 0
```

```
Name        : FirstPet
Description : What was your first pets name?

PS Prompt:\>
```

```
                                                                                          ____________________________/
\________________________________________________________________________________________/ Class [SecurityOptionItem]
  Class [SecurityOptionList] /----------------------------------------------------------------------------------------\
/----------------------------
```

Hosts the output of the list of [SecurityOptionType]'s when thrown through the list of [SecurityOptionItem]'s.

```
    Class SecurityOptionList
    {
        [String]    $Name
        [Object]  $Output
        SecurityOptionList()
        {
            $This.Name = "SecurityOptionList"
            $This.Refresh()
        }
        Clear()
        {
            $This.Output = @( )
        }
        [Object] SecurityOptionItem([String]$Name)
        {
            Return [SecurityOptionItem]::New($Name)
        }
        Add([Object]$Object)
        {
            $This.Output += $Object
        }
        Refresh()
        {
            $This.Clear()
            ForEach ($Name in [System.Enum]::GetNames([SecurityOptionType]))
            {
                $Item             = $This.SecurityOptionItem($Name)
                $Item.Description = Switch ($Item.Index)
                {
                    0 { "What was your first pets name?"                      }
                    1 { "What's the name of the city where you were born?"    }
                    2 { "What was your childhood nickname?"                   }
                    3 { "What's the name of the city where your parents met?" }
                    4 { "What's the first name of your oldest cousin?"        }
                    5 { "What's the name of the first school you attended?"   }
                }

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

The controller class only retrieves the output of this class, which is why it isn't part of the $Ctrl variable.

```
PS Prompt:\> [SecurityOptionList]::New()

Name               Output
----               ------
SecurityOptionList {FirstPet, BirthCity, ChildhoodNick, ParentCity...}

PS Prompt:\>
```

```
                                                                                          ____________________________/
\________________________________________________________________________________________/ Class [SecurityOptionList]
  Class [SecurityOptionSelection] /-----------------------------------------------------------------------------------\
/---------------------------------
```

Returns the selection from the menu, as well as the answer, so that the [VmController] can type the entries into the setup process.

```
Class SecurityOptionSelection
{
    [UInt32]    $Index
    [String]     $Name
    [String] $Question
    [String]   $Answer
    SecurityOptionSelection([UInt32]$Index,[Object]$Item)
    {
        $This.Index    = $Index
        $This.Name     = $Item.Name
        $This.Question = $Item.Description
    }
    SetAnswer([String]$Answer)
    {
        $This.Answer   = $Answer
    }
}
```

```
PS Prompt:\> $Security.Output

Index Name     Question                       Answer
----- ----     --------                       ------
    0 FirstPet What was your first pets name? Whatevs

PS Prompt:\>
```

```
                                                                                                  _________________________________/
\_________________________________________________________________________________________/ Class [SecurityOptionSelection]
  Class [SecurityOptionController] /-----------------------------------------------------------------------------------\
/---------------------------------
```

Controls all of the above [SecurityOption] related classes (*the account/credential stuff is not implemented*).

```
Class SecurityOptionController
{
    [Object]    $Account
    [Object] $Credential
    [Object]       $Slot
    [Object]     $Output
    SecurityOptionController()
    {
        $This.Slot     = $This.SecurityOptionList()
        $This.Clear()
    }
    [Object] SecurityOptionList()
    {
        Return [SecurityOptionList]::New().Output
    }
    [Object] SecurityOptionItem([UInt32]$Index,[String]$Name,[String]$Question)
    {
        Return [SecurityOptionItem]::New($Index,$Name,$Question)
    }
    [Object] SecurityOptionSelection([UInt32]$Index,[Object]$Item)
    {
        Return [SecurityOptionSelection]::New($Index,$Item)
    }
    [String] GetUsername()
    {
        If (!$This.Account)
        {
            Throw "Must insert an account"
        }
        Return "{0}{1}{2}" -f $This.Account.First.Substring(0,1).ToLower(),
                              $This.Account.Last.ToLower(),
                            $This.Account.Year.ToString().Substring(2,2)
    }
    [UInt32] Random()
    {
        Return Get-Random -Max 20
    }
    [String] Char()
    {
        Return "!@#$%^&*(){}[]:;,./\".Substring($This.Random(),1)
    }
    [String] GetPassword()
    {
        $R = $This.Char()
        $H = @{ }
        $H.Add($H.Count,$R)
        $H.Add($H.Count,$This.Account.First.Substring(0,1))
        $H.Add($H.Count,("{0:d2}" -f $This.Account.Month))
        If ($This.Account.MI)
        {
            $H.Add($H.Count,$This.Account.MI)
        }
        $H.Add($H.Count,("{0:d2}" -f $This.Account.Day))
        $H.Add($H.Count,$This.Account.Last.Substring(0,1))
        $H.Add($H.Count,$This.Account.Year.ToString().Substring(2,2))
        $H.Add($H.Count,$R)
        Return $H[0..($H.Count-1)] -join ""
    }
    [PSCredential] PSCredential([String]$Username,[SecureString]$SecureString)
    {
        Return [PSCredential]::New($Username,$SecureString)
    }
    [String] PW()
    {
        If (!$This.Credential)
        {
            Throw "No credential set"
        }
        Return $This.Credential.GetNetworkCredential().Password
    }
    [String] UN()
    {
        If (!$This.Credential)
        {
            Throw "No credential set"
        }
        Return $This.Credential.Username
    }
    SetCredential()
    {
        $SS              = $This.GetPassword() | ConvertTo-SecureString -AsPlainText -Force
        $This.Credential = $This.PSCredential($This.GetUsername(),$SS)
    }
    SetAccount([Object]$Account)
    {
        $This.Account = $Account
    }
    Clear()
    {
        $This.Output = @( )
    }
    Add([UInt32]$Rank,[String]$Answer)
    {
        $Temp = $This.SecurityOptionSelection($This.Output.Count,$This.Slot[$Rank])

        If ($Temp.Name -in $This.Output.Name)
        {
            Throw "Option already selected"
        }
        ElseIf ($Answer -eq "")
        {
            Throw "Cannot have a <null> answer"
        }
        $Temp.SetAnswer($Answer)
        $This.Output += $Temp
    }
}
```

```
PS Prompt:\> $Security

Account Credential Slot                                               Output
------- ---------- ----                                               ------
                   {FirstPet, BirthCity, ChildhoodNick, ParentCity...} {FirstPet}


PS Prompt:\>
```

```
                                                                                                  __________________________________/
\________________________________________________________________________________________________/ Class [SecurityOptionController]
  Class [CountryItem] /----------------------------------------------------------------------------------------------------------------\
/---------------------
```

Literally an item for a country, with a numerical index.

```

    Class CountryItem
    {
        [UInt32] $Index
        [String]  $Name
        CountryItem([UInt32]$Index,[String]$Name)
        {
            $This.Index = $Index
            $This.Name  = $Name
        }
    }

```

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

Index Name
----- ----
    0 Afghanistan

PS Prompt:\>
```

```
                                                                                                                _____________________/
\______________________________________________________________________________________________________________/ Class [CountryItem]
  Class [CountryList] /----------------------------------------------------------------------------------------------------------------\
/---------------------
```

Meant for organizing all available countries during the setup process.
May be implemented into the [Windows 11] setup, however it is [not currently implemented].

```

    Class CountryList
    {
        [UInt32] $Selected
        [Object] $Output
        CountryList()
        {
            $This.Refresh()
        }
        Clear()
        {
            $This.Output = @( )
        }
        [Object] CountryItem([UInt32]$Index,[String]$Name)
        {
            Return [CountryItem]::New($Index,$Name)
        }
        Add([String]$Name)
        {
            $This.Output += $This.CountryItem($This.Output.Count,$Name)
        }
        Select([UInt32]$Index)
        {
            If ($Index -gt $This.Output.Count)
            {
                Throw "Invalid index"
            }

            $This.Selected = $Index
        }
        [Object] Current()
        {
            Return $This.Output[$This.Selected]
        }
        [String[]] Countries()
        {
            Return ("Afghanistan;Åland Islands;Albania;Algeria;American Samoa;"+
            "Andorra;Angola;Anguilla;Antarctica;Antigua and Barbuda;Argentina;"+
            "Armenia;Aruba;Australia;Austrai;Azerbaijan;Bahamas, The;Bahrain;B"+
            "angladesh;Barbados;Belarus;Belgium;Belize;Benin;Bermuda;Bhutan;Bo"+
            "livia;Bonaire, Sint Eustatis and Saba;Bosnia and Herzegovina;Bots"+
            "wana;Bouvet Island;Brazil;British Indian Ocean Territory;British "+
            "Virgin Islands;Brunei;Bulgaria;Burkina Faso;Burundi;Cabo Verde;Ca"+
            "mbodia;Cameroon;Canada;Cayman Islans;Central African Republic;Cha"+
            "d;Chile;China;Christmas Island;Cocos (Keeling) Islands;Colombia;C"+
            "omoros;Congo;Congo (DRC);Cook Islands;Costa Rica;Côte d'Ivoire;Cr"+
            "oatia;Cuba;Curaçao;Cyprus;Czech Republic;Denmark;Djibouti;Dominic"+
            "a;Dominican Republic;Ecuador;Egypt;El Salvador;Equatorial Guinea;"+
            "Eritrea;Estonia;Eswatini;Ethiopia;Falkland Islands;Faroe Islands;"+
            "Fiji;Finland;France;French Guiana;French Polynesia;French Souther"+
            "n Territoes;Gabon;Gambia;Georgia;Germany;Ghana;Gibraltar;Greece;G"+
            "reenland;Grenada;Guadeloupe;Guam;Guatemala;Guernsey;Guinea;Guinea"+
            "-Bissau;Guyana;Haiti;Heard Island and McDonald Islands;Honduras;H"+
            "ong Kong SAR;Hungary;Iceland;India;Indonesia;Iran;Iraq;Ireland;Is"+
            "le of Man;Israel;Italy;Jamaica;Japan;Jersey;Jordan;Kazakhstan;Ken"+
            "ya;Kiribati;Korea;Kosovo;Kuwait;Kyrgyzstan;Laos;Latvia;Lebanon;Le"+
            "sotho;Liberia;Libya;Liechtenstein;Lithuania;Luxembourg;Macao SAR;"+
            "Madagascar;Malawi;Malaysia;Maldives;Mali;Malta;Marshall Islands;M"+
            "artinique;Mauritania;Mauritius;Mayotte;Mexico;Micronesia;Moldova;"+
            "Monaco;Mongolia;Montenegro;Montserrat;Morocco;Mozambique;Myanmar;"+
            "Namibia;Nauru;Nepal;Netherlands;New Caledonia;New Zealand;Nicarag"+
            "ua;Niger;Nigeria;Niue;Norfolk Island;North Korea;North Macedonia;"+
            "Northern Mariana Islands;Norway;Oman;Pakistan;Palau;Palestinian A"+
            "uthority;Panama;Papua New Guinea;Paraguay;Peru;Philippines;Pitcai"+
            "rn Islands;Poland;Portugal;Puerto Rico;Qatar;Reuincion;Romania;Ru"+
            "ssia;Rwanda;Saint Barthélemy;Saint Kiits and Nevis;Saint Lucia;Sa"+
            "int Martin;Saint Pierre and Miquelon;Saint Vincent and the Grenad"+
            "ines;Samoa;San Marino;São Tomé and Príncipe;Saudi Arabia;Senegal;"+
            "Serbia;Seychelles;Sierra Leone;Singapore;Sint Maarten;Slovakia;Sl"+
            "ovenia;Soloman Islands;Somalia;South Africa;South Georgia and the"+
            " South Sandwich Islands;South Sudan;Spain;Sri Lankda;St Kelena, A"+
            "scension and Tristan da Cunha;Sudan;Suriname;Svalbard;Sweden;Swit"+
            "zerland;Syria;Taiwan;Tajikistan;Tanzania;Thailand;Timor-Leste;Tog"+
            "o;Tokelau;Tonga;Trinidad and Tobago;Tunisia;Turkey;Turkmenistan;T"+
            "urks and Caicos Islands;Tuvalu;U.S. Minor Outlying Islands;U.S. V"+
            "irgin Islands;Uganda;Ukraine;United Arab Emirates;United Kingdom;"+
            "United States;Uruguay;Uzbekistan;Vanuatu;Vatican City;Venezuela;V"+
            "ietnam;Wallis and Futuna;Yemen;Zambia;Zimbabwe") -Split ";"
        }
        Refresh()
        {
            $This.Clear()

            ForEach ($Item in $This.Countries())
            {
                $This.Add($Item)
            }

            $This.Selected = $This.Output | ? Name -eq "United States" | % Index
        }
    }
```

```
PS Prompt:\> $Country

Selected Output
-------- ------
     238 {Afghanistan, Åland Islands, Albania, Algeria...}


PS Prompt:\>
```

```
                                                                                                   ______________________/
\_________________________________________________________________________________________________/ Class [CountryList]
  Class [KeyboardItem] /--------------------------------------------------------------------------------------------------\
/----------------------
```

Literally an item meant for a [keyboard (*type/input*)].

```
    Class KeyboardItem
    {
        [UInt32] $Index
        [String]  $Name
        KeyboardItem([UInt32]$Index,[String]$Name)
        {
            $This.Index = $Index
            $This.Name  = $Name
        }
    }
```

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

Index Name
----- ----
    0 US

PS Prompt:\>
```

```
                                                                                                  _______________________/
\________________________________________________________________________________________________/ Class [KeyboardItem]
  Class [KeyboardList] /--------------------------------------------------------------------------------------------------\
/----------------------
```

A list of keyboard items, which is [not currently implemented].

```
    Class KeyboardList
    {
        [UInt32] $Selected
        [Object] $Output
        KeyboardList()
        {
            $This.Refresh()
        }
        Clear()
        {
            $This.Output = @( )
        }
        [Object] KeyboardItem([UInt32]$Index,[String]$Name)
        {
            Return [KeyboardItem]::New($Index,$Name)
        }
        Add([String]$Name)
        {
            $This.Output += $This.KeyboardItem($This.Output.Count,$Name)
        }
        Select([UInt32]$Index)
        {
            If ($Index -gt $This.Output.Count)
            {
                Throw "Invalid index"
            }

            $This.Selected = $Index
        }
        [Object] Current()
        {
            Return $This.Output[$This.Selected]
        }
        [String[]] Keyboards()
        {
            Return ("US;Canadian Multilingual Standard;English (India);Irish;Scottish"+
            " Gaelic;United Kingdom;United States-Dvorak;United States-Dvorak for lef"+
            "t hand;United States-Dvorak for right hand;United States-International;U"+
            "S English Table for IBM Arabic 238_L;Albanian;Azerbaijani (Standard);Aze"+
            "rbaijani Latin;Belgian (Comma);Belgian (Period);Belgian French;Bulgarian"+
            " (Latin);Canadian French;Canadian French (Legacy);Central Atlas Tamazigh"+
            "t;Czech;Czech (QWERTY);Czech Programmers;Danish;Dutch;Estonian;Faeroese;"+
            "Finnish;Finnish with Sami;French;German;German (IBM);Greek (220) Latin;G"+
            "reek (319) Latin;Greek Latin;Greenlandic;Guarani;Hausa;Hawaiian;Hungaria"+
            "n;Hungarian 101-key;Icelandic;Igbo;Inuktitut - Latin;Italian;Italian (14"+
            "2);Japanese;Korean;Latin America;Latvian;Latvian (QWERTY);Latvian (Stand"+
            "ard);Lithuanian;Lithuanian IBM;Lithuanian Standard;Luxembourgish;Maltese"+
            " 47-Key;Maltese 48-Key;Norwegian;Norwegain with Sami;Polish (214);Polish"+
            " (Programmers);Portuguese;Portugese (Brazil ABNT);Portugese (Brazil ABNT"+
            "2);Romanian (Legacy);Romanian (Programmers);Romanian (Standard);Sami Ext"+
            "ended Finland-Sweden;Sami Extended Norway;Serbian (Latin);Sesotho sa Leb"+
            "oa;Setswana;Slovak;Slovak (QWERTY);Slovenian;Sorbian Extended;Sorbian St"+
            "andard;Sorbian Standard (Legacy);Spanish;Spanish Variation;Standard;Swed"+
            "ish;Swedish with Sami;Swiss French;Swiss German;Turkish F;Turkish Q;Turk"+
            "men;United Kingdom Extended;Vietnamese;Wolof;Yoruba") -Split ";"
        }
        Refresh()
        {
            $This.Clear()

            ForEach ($Item in $This.Keyboards())
            {
                $This.Add($Item)
            }

            $This.Selected = $This.Output | ? Name -eq "US" | % Index
        }
    }
```

```
PS Prompt:\> $Keyboard

Selected Output
-------- ------
       0 {US, Canadian Multilingual Standard, English (India), Irish...}

PS Prompt:\>
```

```
                                                                                                 ______________________/
\_______________________________________________________________________________________________/ Class [KeyboardList]
  Class [XamlProperty] /----------------------------------------------------------------------------------------------\
/----------------------
```

Meant for individual [Xaml] properties.

```
    Class XamlProperty
    {
        [UInt32]   $Index
        [String]    $Name
        [Object]    $Type
        [Object] $Control
        XamlProperty([UInt32]$Index,[String]$Name,[Object]$Object)
        {
            $This.Index   = $Index
            $This.Name    = $Name
            $This.Type    = $Object.GetType().Name
            $This.Control = $Object
        }
        [String] ToString()
        {
            Return $This.Name
        }
    }
```

```
PS Prompt:\> $Ctrl.Xaml.Types[0] | Format-List

Index   : 0
Name    : MasterConfig
Type    : DataGrid
Control : System.Windows.Controls.DataGrid Items.Count:1

PS Prompt:\>
```

```
                                                                                                            ______________________/
\___________________________________________________________________________________________/ Class [XamlProperty]
  Class [XamlWindow] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Meant to control the [Xaml] input object, as well as the various controls that instantiate the [window].

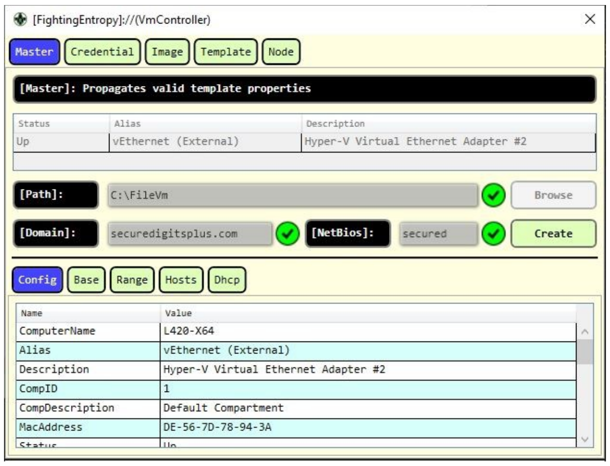
```
    Class XamlWindow
    {
        Hidden [Object]        $Xaml
        Hidden [Object]         $Xml
        [String[]]            $Names
        [Object]              $Types
        [Object]               $Node
        [Object]                 $IO
        [String]          $Exception
        XamlWindow([String]$Xaml)
        {
            If (!$Xaml)
            {
                Throw "Invalid XAML Input"
            }

            [System.Reflection.Assembly]::LoadWithPartialName('presentationframework')

            $This.Xaml           = $Xaml
            $This.Xml            = [XML]$Xaml
            $This.Names          = $This.FindNames()
            $This.Types          = @( )
            $This.Node           = [System.Xml.XmlNodeReader]::New($This.Xml)
            $This.IO             = [System.Windows.Markup.XamlReader]::Load($This.Node)

            ForEach ($X in 0..($This.Names.Count-1))
            {
                $Name            = $This.Names[$X]
                $Object          = $This.IO.FindName($Name)
                $This.IO         | Add-Member -MemberType NoteProperty -Name $Name -Value $Object -Force
                If (!!$Object)
                {
                    $This.Types += $This.XamlProperty($This.Types.Count,$Name,$Object)
                }
            }
        }
        [String[]] FindNames()
        {
            Return [Regex]::Matches($This.Xaml,"( Name\=\`"\w+`")").Value -Replace "( Name=|`")",""
        }
        [Object] XamlProperty([UInt32]$Index,[String]$Name,[Object]$Object)
        {
            Return [XamlProperty]::New($Index,$Name,$Object)
        }
        [Object] Get([String]$Name)
        {
            $Item = $This.Types | ? Name -eq $Name
            If ($Item)
            {
                Return $Item.Control
            }
            Else
            {
                Return $Null
            }
        }
        Invoke()
        {
            Try
            {
                $This.IO.Dispatcher.InvokeAsync({ $This.IO.ShowDialog() }).Wait()
            }
            Catch
            {
                $This.Exception = $PSItem
            }
        }
        [String] ToString()
        {
            Return "<FEModule.XamlWindow[VmControllerXaml]>"
        }
    }
```

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

Names     : {Border, MasterConfig, MasterPath, MasterPathIcon...}
Types     : {MasterConfig, MasterPath, MasterPathIcon, MasterPathBrowse...}
Node      : System.Xml.XmlNodeReader
IO        : System.Windows.Window
Exception :

PS Prompt:\>
```

```
                                                                                                                    ____________________/
\__________________________________________________________________________________________________________/ Class [XamlWindow]
  Class [VmControllerXaml] /--------------------------------------------------------------------------------------------------\
/--------------------------
```

A chunk of [Xaml] that I made in [Visual Studio]. I write it the way I do to [minimize (*text/line*) wrapping].
Same goes for a majority of the code [demonstrated/documented]. Screenshots after the code.

```
    Class VmControllerXaml
    {
        Static [String] $Content = @(
        '<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" ',
        '        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" ',
        '        Title="[FightingEntropy]://(VmController)"',
        '        Height="480"',
        '        Width="640"',
        '        Topmost="True"',
        '        ResizeMode="NoResize"',
        '        Icon="C:\ProgramData\Secure Digits Plus LLC\FightingEntropy\2023.4.0\Graphics\icon.ico"',
        '        HorizontalAlignment="Center"',
        '        WindowStartupLocation="CenterScreen"',
        '        FontFamily="Consolas"',
        '        Background="LightYellow">',
        '    <Window.Resources>',
        '        <Style x:Key="DropShadow">',
        '            <Setter Property="TextBlock.Effect">',
        '                <Setter.Value>',
        '                    <DropShadowEffect ShadowDepth="1"/>',
        '                </Setter.Value>',
        '            </Setter>',
        '        </Style>',
        '        <Style TargetType="ToolTip">',
        '            <Setter Property="Background" Value="#000000"/>',
        '            <Setter Property="Foreground" Value="#66D066"/>',
        '        </Style>',
        '        <Style TargetType="TabItem">',
        '            <Setter Property="Template">',
        '                <Setter.Value>',
        '                    <ControlTemplate TargetType="TabItem">',
        '                        <Border Name="Border"',
        '                                BorderThickness="2"',
        '                                BorderBrush="Black"',
        '                                CornerRadius="5"',
        '                                Margin="2">',
        '                            <ContentPresenter x:Name="ContentSite"',
        '                                              VerticalAlignment="Center"',
        '                                              HorizontalAlignment="Right"',
        '                                              ContentSource="Header"',
        '                                              Margin="5"/>',
        '                        </Border>',
        '                        <ControlTemplate.Triggers>',
        '                            <Trigger Property="IsSelected" ',
        '                                     Value="True">',
        '                                <Setter TargetName="Border" ',
        '                                        Property="Background" ',
        '                                        Value="#4444FF"/>',
        '                                <Setter Property="Foreground" ',
        '                                        Value="#FFFFFF"/>',
        '                            </Trigger>',
        '                            <Trigger Property="IsSelected" ',
        '                                     Value="False">',
        '                                <Setter TargetName="Border" ',
        '                                        Property="Background" ',
        '                                        Value="#DFFFBA"/>',
        '                                <Setter Property="Foreground" ',
        '                                        Value="#000000"/>',
        '                            </Trigger>',
        '                        </ControlTemplate.Triggers>',
        '                    </ControlTemplate>',
        '                </Setter.Value>',
        '            </Setter>',
        '        </Style>',
        '        <Style TargetType="Button">',
        '            <Setter Property="Margin" Value="5"/>',
        '            <Setter Property="Padding" Value="5"/>',
        '            <Setter Property="FontWeight" Value="Heavy"/>',
        '            <Setter Property="Foreground" Value="Black"/>',
        '            <Setter Property="Background" Value="#DFFFBA"/>',
        '            <Setter Property="BorderThickness" Value="2"/>',
        '            <Setter Property="VerticalContentAlignment" Value="Center"/>',
        '            <Style.Resources>',
        '                <Style TargetType="Border">',
        '                    <Setter Property="CornerRadius" Value="5"/>',
        '                </Style>',
        '            </Style.Resources>',
        '        </Style>',
        '        <Style TargetType="{x:Type TextBox}" BasedOn="{StaticResource DropShadow}">',
        '            <Setter Property="TextBlock.TextAlignment" Value="Left"/>',
        '            <Setter Property="VerticalContentAlignment" Value="Center"/>',
        '            <Setter Property="HorizontalContentAlignment" Value="Left"/>',
        '            <Setter Property="Height" Value="24"/>',
        '            <Setter Property="Margin" Value="4"/>',
        '            <Setter Property="FontSize" Value="12"/>',
        '            <Setter Property="Foreground" Value="#000000"/>',
        '            <Setter Property="TextWrapping" Value="Wrap"/>',
        '            <Style.Resources>',
        '                <Style TargetType="Border">',
        '                    <Setter Property="CornerRadius" Value="2"/>',
        '                </Style>',
        '            </Style.Resources>',
        '        </Style>',
        '        <Style TargetType="{x:Type PasswordBox}" BasedOn="{StaticResource DropShadow}">',
        '            <Setter Property="TextBlock.TextAlignment" Value="Left"/>',
        '            <Setter Property="VerticalContentAlignment" Value="Center"/>',
        '            <Setter Property="HorizontalContentAlignment" Value="Left"/>',
        '            <Setter Property="Margin" Value="4"/>',
        '            <Setter Property="Height" Value="24"/>',
        '            <Style.Resources>',
        '                <Style TargetType="Border">',
        '                    <Setter Property="CornerRadius" Value="2"/>',
        '                </Style>',
        '            </Style.Resources>',
        '        </Style>',
        '        <Style TargetType="ComboBox">',
        '            <Setter Property="Height" Value="24"/>',
        '            <Setter Property="Margin" Value="5"/>',
        '            <Setter Property="FontSize" Value="12"/>',
        '            <Setter Property="FontWeight" Value="Normal"/>',
        '        </Style>',
        '        <Style TargetType="CheckBox">',
        '            <Setter Property="VerticalContentAlignment" Value="Center"/>',
        '        </Style>',
        '        <Style TargetType="DataGrid">',
        '            <Setter Property="Margin" ',
        '                    Value="5"/>',
        '            <Setter Property="AutoGenerateColumns"',
        '                    Value="False"/>',
        '            <Setter Property="AlternationCount"',
        '                    Value="2"/>',
        '            <Setter Property="HeadersVisibility"',
        '                    Value="Column"/>',
        '            <Setter Property="CanUserResizeRows"',
        '                    Value="False"/>',
        '            <Setter Property="CanUserAddRows"',
        '                    Value="False"/>',
        '            <Setter Property="IsReadOnly"',
        '                    Value="True"/>',
        '            <Setter Property="IsTabStop"',
        '                    Value="True"/>',
        '            <Setter Property="IsTextSearchEnabled"',
        '                    Value="True"/>',
        '            <Setter Property="SelectionMode"',
        '                    Value="Single"/>',
        '            <Setter Property="ScrollViewer.CanContentScroll"',
        '                    Value="True"/>',
        '            <Setter Property="ScrollViewer.VerticalScrollBarVisibility"',
        '                    Value="Auto"/>',
        '            <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"',
        '                    Value="Auto"/>',
        '        </Style>',
        '        <Style TargetType="DataGridRow">',
        '            <Setter Property="VerticalAlignment"',
        '                    Value="Center"/>',
        '            <Setter Property="VerticalContentAlignment"',
        '                    Value="Center"/>',
        '            <Setter Property="TextBlock.VerticalAlignment"',
        '                    Value="Center"/>',
        '            <Setter Property="Height" Value="20"/>',
        '            <Setter Property="FontSize" Value="12"/>',
        '            <Style.Triggers>',
        '                <Trigger Property="AlternationIndex" ',
        '                         Value="0">',
        '                    <Setter Property="Background" ',
        '                            Value="White"/>',
        '                </Trigger>',
        '                <Trigger Property="AlternationIndex" Value="1">',
        '                    <Setter Property="Background" ',
        '                            Value="#FFD6FFFB"/>',
        '                </Trigger>',
        '                <Trigger Property="IsMouseOver" Value="True">',
        '                    <Setter Property="ToolTip">',
        '                        <Setter.Value>',
        '                            <TextBlock TextWrapping="Wrap" ',
        '                                       Width="400" ',
        '                                       Background="#000000" ',
        '                                       Foreground="#00FF00"/>',
        '                        </Setter.Value>',
        '                    </Setter>',
        '                    <Setter Property="ToolTipService.ShowDuration" Value="360000000"/>',
        '                </Trigger>',
        '            </Style.Triggers>',
        '        </Style>',
        '        <Style TargetType="DataGridColumnHeader">',
        '            <Setter Property="FontSize"   Value="10"/>',
        '            <Setter Property="FontWeight" Value="Normal"/>',
        '        </Style>',
        '        <Style TargetType="TabControl">',
        '            <Setter Property="TabStripPlacement" Value="Top"/>',
        '            <Setter Property="HorizontalContentAlignment" Value="Center"/>',
        '            <Setter Property="Background" Value="LightYellow"/>',
        '        </Style>',
        '        <Style TargetType="GroupBox">',
        '            <Setter Property="Foreground" Value="Black"/>',
        '            <Setter Property="Margin" Value="5"/>',
        '            <Setter Property="FontSize" Value="12"/>',
        '            <Setter Property="FontWeight" Value="Normal"/>',
        '        </Style>',
        '        <Style TargetType="Label">',
        '            <Setter Property="Margin" Value="5"/>',
        '            <Setter Property="FontWeight" Value="Bold"/>',
        '            <Setter Property="Background" Value="Black"/>',
        '            <Setter Property="Foreground" Value="White"/>',
        '            <Setter Property="BorderBrush" Value="Gray"/>',
        '            <Setter Property="BorderThickness" Value="2"/>',
        '            <Style.Resources>',
        '                <Style TargetType="Border">',
        '                    <Setter Property="CornerRadius" Value="5"/>',
        '                </Style>',
        '            </Style.Resources>',
        '        </Style>',
        '        <Style x:Key="LabelGray" TargetType="Label">',
        '            <Setter Property="Margin" Value="5"/>',
        '            <Setter Property="FontWeight" Value="Bold"/>',
        '            <Setter Property="Background" Value="DarkSlateGray"/>',
        '            <Setter Property="Foreground" Value="White"/>',
        '            <Setter Property="BorderBrush" Value="Black"/>',
        '            <Setter Property="BorderThickness" Value="2"/>',
        '            <Setter Property="HorizontalContentAlignment" Value="Center"/>',
        '            <Style.Resources>',
        '                <Style TargetType="Border">',
        '                    <Setter Property="CornerRadius" Value="5"/>',
        '                </Style>',
        '            </Style.Resources>',
        '        </Style>',
        '        <Style x:Key="LabelRed" TargetType="Label">',
        '            <Setter Property="Margin" Value="5"/>',
        '            <Setter Property="FontWeight" Value="Bold"/>',
        '            <Setter Property="Background" Value="IndianRed"/>',
        '            <Setter Property="Foreground" Value="White"/>',
        '            <Setter Property="BorderBrush" Value="Black"/>',
        '            <Setter Property="BorderThickness" Value="2"/>',
        '            <Setter Property="HorizontalContentAlignment" Value="Left"/>',
        '            <Style.Resources>',
        '                <Style TargetType="Border">',
        '                    <Setter Property="CornerRadius" Value="5"/>',
        '                </Style>',
        '            </Style.Resources>',
        '        </Style>',
        '        <Style x:Key="Line" TargetType="Border">',
        '            <Setter Property="Background" Value="Black"/>',
        '            <Setter Property="BorderThickness" Value="0"/>',
        '            <Setter Property="Margin" Value="4"/>',
        '        </Style>',
        '    </Window.Resources>',
        '    <TabControl Grid.Row="0">',
        '        <TabItem Header="Master">',
        '            <Grid>',
        '                <Grid.RowDefinitions>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="70"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="10"/>',
        '                    <RowDefinition Height="*"/>',
        '                </Grid.RowDefinitions>',
        '                <Label Content="[Master]: Propagates valid template properties"/>',
        '                <DataGrid Grid.Row="1" Name="MasterConfig">',
        '                    <DataGrid.Columns>',
        '                        <DataGridTextColumn Header="Status"',
        '                                            Binding="{Binding Status}"',
        '                                            Width="100"/>',
        '                        <DataGridTextColumn Header="Alias"',
        '                                            Binding="{Binding Alias}"',
        '                                            Width="200"/>',
        '                        <DataGridTextColumn Header="Description"',
        '                                            Binding="{Binding Description}"',
        '                                            Width="*"/>',
        '                    </DataGrid.Columns>',
        '                </DataGrid>',
        '                <Grid Grid.Row="2">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="100"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="25"/>',
        '                        <ColumnDefinition Width="100"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Label   Grid.Column="0"',
        '                             Content="[Path]:"/>',
        '                    <TextBox Grid.Column="1"',
        '                             Name="MasterPath"/>',
        '                    <Image   Grid.Column="2"',
        '                             Name="MasterPathIcon"/>',
        '                    <Button  Grid.Column="3"',
        '                             Name="MasterPathBrowse"',
        '                             Content="Browse"/>',
        '                </Grid>',
        '                <Grid Grid.Row="3">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="100"/>',
        '                        <ColumnDefinition Width="2*"/>',
        '                        <ColumnDefinition Width="25"/>',
        '                        <ColumnDefinition Width="100"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="25"/>',
        '                        <ColumnDefinition Width="100"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Label   Grid.Column="0" Content="[Domain]:"/>',
        '                    <TextBox Grid.Column="1" Name="MasterDomain"/>',
        '                    <Image   Grid.Column="2" Name="MasterDomainIcon"/>',
        '                    <Label   Grid.Column="3" Content="[NetBios]:"/>',
        '                    <TextBox Grid.Column="4" Name="MasterNetBios"/>',
        '                    <Image   Grid.Column="5" Name="MasterNetBiosIcon"/>',
        '                    <Button  Grid.Column="7" Name="MasterCreate" Content="Create"/>',
        '                </Grid>',
        '                <Border Grid.Row="4" Background="Black" Margin="4"/>',
        '                <TabControl Grid.Row="5">',
        '                    <TabItem Header="Config">',
        '                        <DataGrid Name="MasterConfigOutput">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Name"',
        '                                                    Binding="{Binding Name}"',
        '                                                    Width="150"/>',
        '                                <DataGridTextColumn Header="Value"',
        '                                                    Binding="{Binding Value}"',
        '                                                    Width="*"/>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                    </TabItem>',
        '                    <TabItem Header="Base">',
        '                        <DataGrid Name="MasterBase">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Name"',
        '                                                    Binding="{Binding Name}"',
        '                                                    Width="150"/>',
        '                                <DataGridTextColumn Header="Value"',
        '                                                    Binding="{Binding Value}"',
        '                                                    Width="*"/>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                    </TabItem>',
        '                    <TabItem Header="Range">',
        '                        <DataGrid Name="MasterRange">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Index"',
        '                                                    Binding="{Binding Index}"',
        '                                                    Width="50"/>',
        '                                <DataGridTextColumn Header="Count"',
        '                                                    Binding="{Binding Count}"',
        '                                                    Width="100"/>',
        '                                <DataGridTextColumn Header="Netmask"',
        '                                                    Binding="{Binding Netmask}"',
        '                                                    Width="150"/>',
        '                                <DataGridTextColumn Header="Notation"',
        '                                                    Binding="{Binding Notation}"',
        '                                                    Width="*"/>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                    </TabItem>',
        '                    <TabItem Header="Hosts">',
        '                        <DataGrid Name="MasterHosts">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Index"',
        '                                                    Binding="{Binding Index}"',
        '                                                    Width="50"/>',
        '                                <DataGridTemplateColumn Header="Status" Width="45">',
        '                                    <DataGridTemplateColumn.CellTemplate>',
        '                                        <DataTemplate>',
        '                                            <ComboBox SelectedIndex="{Binding Status}"',
        '                                                      Margin="0"',
        '                                                      Padding="2"',
        '                                                      Height="18"',
        '                                                      FontSize="10"',
        '                                                      VerticalContentAlignment="Center">',
        '                                                <ComboBoxItem Content="[-]"/>',
        '                                                <ComboBoxItem Content="[+]"/>',
        '                                            </ComboBox>',
        '                                        </DataTemplate>',
        '                                    </DataGridTemplateColumn.CellTemplate>',
        '                                </DataGridTemplateColumn>',
        '                                <DataGridTextColumn Header="Type"',
        '                                                    Binding="{Binding Type}"',
        '                                                    Width="80"/>',
        '                                <DataGridTextColumn Header="IpAddress"',
        '                                                    Binding="{Binding IpAddress}"',
        '                                                    Width="120"/>',
        '                                <DataGridTextColumn Header="Hostname"',
        '                                                    Binding="{Binding Hostname}"',
        '                                                    Width="*"/>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                    </TabItem>',
        '                    <TabItem Header="Dhcp">',
        '                        <DataGrid Name="MasterDhcp">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Name"',
        '                                                    Binding="{Binding Name}"',
        '                                                    Width="150"/>',
        '                                <DataGridTextColumn Header="Value"',
        '                                                    Binding="{Binding Value}"',
        '                                                    Width="*"/>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                    </TabItem>',
        '                </TabControl>',
        '            </Grid>',
        '        </TabItem>',
        '        <TabItem Header="Credential">',
        '            <Grid>',
        '                <Grid.RowDefinitions>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="110"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="10"/>',
        '                    <RowDefinition Height="*"/>',
        '                </Grid.RowDefinitions>',
        '                <Label Content="[Credential]: Creates (standard/add&apos;l) credential(s)"/>',
        '                <DataGrid Grid.Row="1" Name="CredentialOutput">',
        '                    <DataGrid.Columns>',
        '                        <DataGridTextColumn Header="Type"',
        '                                            Binding="{Binding Type}"',
        '                                            Width="90"/>',
        '                        <DataGridTextColumn Header="Username"',
        '                                            Binding="{Binding Username}"',
        '                                            Width="*"/>',
        '                        <DataGridTextColumn Header="Password"',
        '                                            Binding="{Binding Pass}"',
        '                                            Width="150"/>',
        '                    </DataGrid.Columns>',
        '                </DataGrid>',
        '                <Grid Grid.Row="2">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Button Grid.Column="0"',
        '                            Name="CredentialCreate"',
        '                            Content="Create"/>',
        '                    <Button Grid.Column="1"',
        '                            Name="CredentialRemove"',
        '                            Content="Remove"/>',
        '                </Grid>',
        '                <Border Grid.Row="3" Background="Black" Margin="4"/>',
        '                <Grid Grid.Row="4">',
        '                    <Grid.RowDefinitions>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                    </Grid.RowDefinitions>',
        '                    <Grid Grid.Row="0">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="150"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Label    Grid.Column="0" Content="[Type]:"/>',
        '                        <ComboBox Grid.Column="1"',
        '                                  Name="CredentialType"',
        '                                  SelectedIndex="0">',
        '                            <ComboBoxItem Content="Setup"/>',
        '                            <ComboBoxItem Content="System"/>',
        '                            <ComboBoxItem Content="Service"/>',
        '                            <ComboBoxItem Content="User"/>',
        '                            <ComboBoxItem Content="Microsoft"/>',
        '                        </ComboBox>',
        '                        <DataGrid Grid.Column="2"',
        '                                  HeadersVisibility="None"',
        '                                  Name="CredentialDescription"',
        '                                  Margin="10">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Description"',
        '                                                    Binding="{Binding Description}"',
        '                                                    Width="*"/>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                    </Grid>',
        '                    <Grid Grid.Row="1">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="300"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Label Grid.Column="0" Content="[Username]:"/>',
        '                        <TextBox Grid.Column="1"',
        '                                 Name="CredentialUsername"/>',
        '                        <Image Grid.Column="2" Name="CredentialUsernameIcon"/>',
        '                    </Grid>',
        '                    <Grid Grid.Row="2">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="300"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Label Grid.Column="0" Content="[Password]:"/>',
        '                        <PasswordBox Grid.Column="1"',
        '                                 Name="CredentialPassword"/>',
        '                        <Image Grid.Column="2" Name="CredentialPasswordIcon"/>',
        '                    </Grid>',
        '                    <Grid Grid.Row="3">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="300"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Label Grid.Column="0" Content="[Confirm]:"/>',
        '                        <PasswordBox Grid.Column="1"',
        '                                     Name="CredentialConfirm"/>',
        '                        <Image Grid.Column="2" Name="CredentialConfirmIcon"/>',
        '                        <Button  Grid.Column="3"',
        '                                 Name="CredentialGenerate"',
        '                                 Content="Generate"/>',
        '                    </Grid>',
        '                    <Grid Grid.Row="4">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="300"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Label Grid.Column="0" Content="[Pin]:"/>',
        '                        <PasswordBox Grid.Column="1"',
        '                                     Name="CredentialPin"/>',
        '                        <Image Grid.Column="2" Name="CredentialPinIcon"/>',
        '                    </Grid>',
        '                </Grid>',
        '            </Grid>',
        '        </TabItem>',
        '        <TabItem Header="Image">',
        '            <Grid>',
        '                <Grid.RowDefinitions>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="110"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="10"/>',
        '                    <RowDefinition Height="*"/>',
        '                </Grid.RowDefinitions>',
        '                <Label Grid.Row="0" ',
        '                       Content="[Image]: Load images for templates to utilize"/>',
        '                <DataGrid Grid.Row="1" Name="ImageStore">',
        '                    <DataGrid.Columns>',
        '                        <DataGridTextColumn Header="Index"',
        '                                            Binding="{Binding Index}"',
        '                                            Width="40"/>',
        '                        <DataGridTextColumn Header="Type"',
        '                                            Binding="{Binding Type}"',
        '                                            Width="90"/>',
        '                        <DataGridTextColumn Header="Version"',
        '                                            Binding="{Binding Version}"',
        '                                            Width="110"/>',
        '                        <DataGridTextColumn Header="Name"',
        '                                            Binding="{Binding Name}"',
        '                                            Width="500"/>',
        '                    </DataGrid.Columns>',
        '                </DataGrid>',
        '                <Grid Grid.Row="2">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="100"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="25"/>',
        '                        <ColumnDefinition Width="100"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Button  Grid.Column="0"',
        '                             Name="ImageImport"',
        '                             Content="Import"/>',
        '                    <TextBox Grid.Column="1"',
        '                             Name="ImagePath"/>',
        '                    <Image   Grid.Column="2"',
        '                             Name="ImagePathIcon"/>',
        '                    <Button  Grid.Column="3"',
        '                             Name="ImagePathBrowse"',
        '                             Content="Browse"/>',
        '                </Grid>',
        '                <Border Grid.Row="3" Background="Black" Margin="4"/>',
        '                <DataGrid Grid.Row="4" Name="ImageStoreContent">',
        '                    <DataGrid.Columns>',
        '                        <DataGridTextColumn Header="Index"',
        '                                            Binding="{Binding Index}"',
        '                                            Width="40"/>',
        '                        <DataGridTextColumn Header="Name"',
        '                                            Binding="{Binding DestinationName}"',
        '                                            Width="300"/>',
        '                        <DataGridTextColumn Header="Size"',
        '                                            Binding="{Binding Size}"',
        '                                            Width="80"/>',
        '                        <DataGridTextColumn Header="Label"',
        '                                            Binding="{Binding Label}"',
        '                                            Width="*"/>',
        '                    </DataGrid.Columns>',
        '                </DataGrid>',
        '            </Grid>',
        '        </TabItem>',
        '        <TabItem Header="Template">',
        '            <Grid>',
        '                <Grid.RowDefinitions>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="110"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="10"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="40"/>',
        '                </Grid.RowDefinitions>',
        '                <Label Content="[Template]: Generates VM template(s) for [Hyper-V]"/>',
        '                <DataGrid Grid.Row="1"',
        '                              Name="TemplateOutput"',
        '                              ScrollViewer.CanContentScroll="True"',
        '                              ScrollViewer.VerticalScrollBarVisibility="Auto"',
        '                              ScrollViewer.HorizontalScrollBarVisibility="Visible">',
        '                    <DataGrid.Columns>',
        '                        <DataGridTextColumn Header="Index"',
        '                                            Binding="{Binding Index}"',
        '                                            Width="40"/>',
        '                        <DataGridTextColumn Header="Name"',
        '                                            Binding="{Binding Name}"',
        '                                            Width="100"/>',
        '                        <DataGridTextColumn Header="Role"',
        '                                            Binding="{Binding Role}"',
        '                                            Width="60"/>',
        '                        <DataGridTextColumn Header="Memory"',
        '                                            Binding="{Binding Memory.Size}"',
        '                                            Width="60"/>',
        '                        <DataGridTextColumn Header="Hdd"',
        '                                            Binding="{Binding Hdd.Size}"',
        '                                            Width="60"/>',
        '                        <DataGridTextColumn Header="Gen"',
        '                                            Binding="{Binding Gen}"',
        '                                            Width="40"/>',
        '                        <DataGridTextColumn Header="Core"',
        '                                            Binding="{Binding Core}"',
        '                                            Width="40"/>',
        '                        <DataGridTextColumn Header="SwitchId"',
        '                                            Binding="{Binding SwitchId}"',
        '                                            Width="100"/>',
        '                        <DataGridTextColumn Header="Image"',
        '                                            Binding="{Binding Image}"',
        '                                            Width="350"/>',
        '                    </DataGrid.Columns>',
        '                </DataGrid>',
        '                <Grid Grid.Row="2">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Button Grid.Column="0"',
        '                            Content="Create"',
        '                            Name="TemplateCreate"/>',
        '                    <Button Grid.Column="1"',
        '                            Content="Remove"',
        '                            Name="TemplateRemove"/>',
        '                    <Button Grid.Column="2"',
        '                            Content="Export"',
        '                            Name="TemplateExport"/>',
        '                </Grid>',
        '                <Border Grid.Row="3" Background="Black" Margin="4"/>',
        '                <Grid Grid.Row="4">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="100"/>',
        '                        <ColumnDefinition Width="120"/>',
        '                        <ColumnDefinition Width="25"/>',
        '                        <ColumnDefinition Width="100"/>',
        '                        <ColumnDefinition Width="120"/>',
        '                        <ColumnDefinition Width="120"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Label Grid.Column="0" Content="[Name]:"/>',
        '                    <TextBox Grid.Column="1" Name="TemplateName"/>',
        '                    <Image Grid.Column="2" Name="TemplateNameIcon"/>',
        '                    <Label Grid.Column="3" Content="[Role]:"/>',
        '                    <ComboBox Grid.Column="4" Name="TemplateRole">',
        '                        <ComboBoxItem Content="Server"/>',
        '                        <ComboBoxItem Content="Client"/>',
        '                       <ComboBoxItem Content="Unix"/>',
        '                   </ComboBox>',
        '                   <Label Grid.Column="5" Content="[Credentials]:"/>',
        '                   <TextBox Grid.Column="6"',
        '                            Name="TemplateCredentialCount"',
        '                            IsReadOnly="True"/>',
        '               </Grid>',
        '               <Grid Grid.Row="5">',
        '                   <Grid.ColumnDefinitions>',
        '                       <ColumnDefinition Width="100"/>',
        '                       <ColumnDefinition Width="*"/>',
        '                       <ColumnDefinition Width="25"/>',
        '                       <ColumnDefinition Width="90"/>',
        '                   </Grid.ColumnDefinitions>',
        '                   <Label   Grid.Column="0"',
        '                            Content="[Path]:"/>',
        '                   <TextBox Grid.Column="1"',
        '                            Name="TemplatePath"',
        '                            Text="&lt;Select a path&gt;"/>',
        '                   <Image   Grid.Column="2"',
        '                            Name="TemplatePathIcon"/>',
        '                   <Button  Grid.Column="3"',
        '                            Name="TemplatePathBrowse"',
        '                            Content="Browse"/>',
        '               </Grid>',
        '               <Grid Grid.Row="6">',
        '                   <Grid.ColumnDefinitions>',
        '                       <ColumnDefinition Width="105"/>',
        '                       <ColumnDefinition Width="50"/>',
        '                       <ColumnDefinition Width="95"/>',
        '                       <ColumnDefinition Width="*"/>',
        '                       <ColumnDefinition Width="110"/>',
        '                       <ColumnDefinition Width="50"/>',
        '                       <ColumnDefinition Width="95"/>',
        '                       <ColumnDefinition Width="50"/>',
        '                   </Grid.ColumnDefinitions>',
        '                   <Label    Grid.Column="0"',
        '                             Content="[Memory/GB]:"',
        '                             Style="{StaticResource LabelRed}"/>',
        '                   <ComboBox Grid.Column="1" ',
        '                             Name="TemplateMemory"',
        '                             SelectedIndex="0">',
        '                       <ComboBoxItem Content="2"/>',
        '                       <ComboBoxItem Content="4"/>',
        '                       <ComboBoxItem Content="8"/>',
        '                       <ComboBoxItem Content="16"/>',
        '                   </ComboBox>',
        '                   <Label Grid.Column="2"',
        '                          Content="[Drive/GB]:"',
        '                          Style="{StaticResource LabelRed}"/>',
        '                   <ComboBox Grid.Column="3"',
        '                             Name="TemplateHardDrive"',
        '                             SelectedIndex="1">',
        '                       <ComboBoxItem Content="32"/>',
        '                       <ComboBoxItem Content="64"/>',
        '                       <ComboBoxItem Content="128"/>',
        '                       <ComboBoxItem Content="256"/>',
        '                   </ComboBox>',
        '                   <Label Grid.Column="4"',
        '                          Content="[Generation]:"',
        '                          Style="{StaticResource LabelRed}"/>',
        '                   <ComboBox Grid.Column="5"',
        '                             Name="TemplateGeneration"',
        '                             SelectedIndex="1">',
        '                       <ComboBoxItem Content="1"/>',
        '                       <ComboBoxItem Content="2"/>',
        '                   </ComboBox>',
        '                   <Label Grid.Column="6"',
        '                          Content="[CPU/Core]:"',
        '                          Style="{StaticResource LabelRed}"/>',
        '                   <ComboBox Grid.Column="7"',
        '                             Name="TemplateCore"',
        '                              SelectedIndex="1">',
        '                        <ComboBoxItem Content="1"/>',
        '                        <ComboBoxItem Content="2"/>',
        '                        <ComboBoxItem Content="3"/>',
        '                        <ComboBoxItem Content="4"/>',
        '                    </ComboBox>',
        '                </Grid>',
        '                <Grid Grid.Row="7">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="105"/>',
        '                        <ColumnDefinition Width="150"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Label     Grid.Column="0" Content="[Switch]:"/>',
        '                    <ComboBox  Grid.Column="1" Name="TemplateSwitch"/>',
        '                    <TextBlock Grid.Column="2"',
        '                               Foreground="Black"',
        '                               VerticalAlignment="Center"',
        '                               Text="[Virtual switch to use]"/>',
        '                </Grid>',
        '                <Grid Grid.Row="8">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="105"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="25"/>',
        '                        <ColumnDefinition Width="90"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Label Grid.Column="0" Content="[Image/Iso]:"/>',
        '                    <TextBox Grid.Column="1" ',
        '                             Name="TemplateImagePath"',
        '                             Text="&lt;Select an image&gt;"/>',
        '                    <Image   Grid.Column="2"',
        '                             Name="TemplateImagePathIcon"/>',
        '                    <Button  Grid.Column="3"',
        '                             Name="TemplateImagePathBrowse"',
        '                             Content="Browse"/>',
        '                </Grid>',
        '            </Grid>',
        '        </TabItem>',
        '        <TabItem Header="Node" Height="32" VerticalAlignment="Top">',
        '            <Grid>',
        '                <Grid.RowDefinitions>',
        '                    <RowDefinition Height="40"/>',
        '                    <RowDefinition Height="*"/>',
        '                </Grid.RowDefinitions>',
        '                <Grid Grid.Row="0">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="120"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Label Grid.Column="0"',
        '                       Content="[Node]: Manages switches, hosts, and templates"/>',
        '                    <ComboBox Grid.Column="1" Name="NodeSlot" SelectedIndex="1">',
        '                        <ComboBoxItem Content="Switch(es)"/>',
        '                        <ComboBoxItem Content="Host(s)"/>',
        '                    </ComboBox>',
        '                </Grid>',
        '                <Grid Grid.Row="1" Name="NodeSwitchPanel" Visibility="Collapsed">',
        '                    <Grid>',
        '                        <Grid.RowDefinitions>',
        '                            <RowDefinition Height="110"/>',
        '                            <RowDefinition Height="40"/>',
        '                            <RowDefinition Height="10"/>',
        '                            <RowDefinition Height="40"/>',
        '                        </Grid.RowDefinitions>',
        '                        <DataGrid Grid.Row="0" Name="NodeSwitch">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Index"',
        '                                                        Binding="{Binding Index}"',
        '                                                        Width="50"/>',
        '                                <DataGridTextColumn Header="Name"',
        '                                                        Binding="{Binding Name}"',
        '                                                       Width="125"/>',
        '                               <DataGridTextColumn Header="Type"',
        '                                                       Binding="{Binding Type}"',
        '                                                       Width="100"/>',
        '                               <DataGridTextColumn Header="Description"',
        '                                                       Binding="{Binding Description}"',
        '                                                       Width="*"/>',
        '                           </DataGrid.Columns>',
        '                       </DataGrid>',
        '                       <Grid Grid.Row="1">',
        '                           <Grid.ColumnDefinitions>',
        '                               <ColumnDefinition Width="*"/>',
        '                               <ColumnDefinition Width="*"/>',
        '                               <ColumnDefinition Width="*"/>',
        '                           </Grid.ColumnDefinitions>',
        '                           <Button Grid.Column="0"',
        '                                       Content="Create"',
        '                                       Name="NodeSwitchCreate"/>',
        '                           <Button Grid.Column="1"',
        '                                       Content="Remove"',
        '                                       Name="NodeSwitchRemove"/>',
        '                           <Button Grid.Column="2"',
        '                                       Content="Update"',
        '                                       Name="NodeSwitchUpdate"/>',
        '                       </Grid>',
        '                       <Border Grid.Row="2" Background="Black" Margin="4"/>',
        '                       <Grid Grid.Row="3">',
        '                           <Grid.ColumnDefinitions>',
        '                               <ColumnDefinition Width="100"/>',
        '                               <ColumnDefinition Width="*"/>',
        '                               <ColumnDefinition Width="25"/>',
        '                               <ColumnDefinition Width="100"/>',
        '                               <ColumnDefinition Width="100"/>',
        '                           </Grid.ColumnDefinitions>',
        '                           <Label    Grid.Column="0" Content="[Name]:"/>',
        '                           <TextBox  Grid.Column="1" Name="NodeSwitchName"/>',
        '                           <Image    Grid.Column="2" Name="NodeSwitchIcon"/>',
        '                           <Label    Grid.Column="3" Content="[Type]:"/>',
        '                           <ComboBox Grid.Column="4" Name="NodeSwitchType" SelectedIndex="0">',
        '                               <ComboBoxItem Content="External"/>',
        '                               <ComboBoxItem Content="Internal"/>',
        '                               <ComboBoxItem Content="Private"/>',
        '                           </ComboBox>',
        '                       </Grid>',
        '                   </Grid>',
        '               </Grid>',
        '               <Grid Grid.Row="1" Name="NodeHostPanel" Visibility="Visible">',
        '                   <Grid>',
        '                       <Grid.RowDefinitions>',
        '                           <RowDefinition Height="110"/>',
        '                           <RowDefinition Height="40"/>',
        '                           <RowDefinition Height="10"/>',
        '                           <RowDefinition Height="*"/>',
        '                           <RowDefinition Height="40"/>',
        '                       </Grid.RowDefinitions>',
        '                       <DataGrid Grid.Row="0" Name="NodeHost">',
        '                           <DataGrid.Columns>',
        '                               <DataGridTextColumn Header="Index"',
        '                                           Binding="{Binding Index}"',
        '                                           Width="40"/>',
        '                               <DataGridTextColumn Header="Guid"',
        '                                           Binding="{Binding Guid}"',
        '                                           Width="350"/>',
        '                               <DataGridTextColumn Header="Name"',
        '                                           Binding="{Binding Name}"',
        '                                           Width="*"/>',
        '                               <DataGridTextColumn Header="Type"',
        '                                           Binding="{Binding Type}"',
        '                                           Width="100"/>',
        '                           </DataGrid.Columns>',
        '                       </DataGrid>',
        '                       <Grid Grid.Row="1">',
        '                            <Grid.ColumnDefinitions>',
        '                                <ColumnDefinition Width="*"/>',
        '                                <ColumnDefinition Width="*"/>',
        '                                <ColumnDefinition Width="*"/>',
        '                            </Grid.ColumnDefinitions>',
        '                            <Button Grid.Column="0"',
        '                                        Content="Create"',
        '                                        Name="NodeHostCreate"/>',
        '                            <Button Grid.Column="1"',
        '                                        Content="Remove"',
        '                                        Name="NodeHostRemove"/>',
        '                            <Button Grid.Column="2"',
        '                                        Content="Update"',
        '                                        Name="NodeHostUpdate"/>',
        '                        </Grid>',
        '                        <Border Grid.Row="2" Background="Black" Margin="4"/>',
        '                        <DataGrid Grid.Row="3" Name="NodeHostExtension">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Name"',
        '                                                    Binding="{Binding Name}"',
        '                                                    Width="150"/>',
        '                                <DataGridTextColumn Header="Value"',
        '                                                    Binding="{Binding Value}"',
        '                                                    Width="*"/>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                        <Grid Grid.Row="4">',
        '                            <Grid.ColumnDefinitions>',
        '                                <ColumnDefinition Width="100"/>',
        '                                <ColumnDefinition Width="*"/>',
        '                                <ColumnDefinition Width="25"/>',
        '                                <ColumnDefinition Width="100"/>',
        '                            </Grid.ColumnDefinitions>',
        '                            <Button  Grid.Column="0"',
        '                                         Content="Import"',
        '                                         Name="NodeTemplateImport"/>',
        '                            <TextBox Grid.Column="1"',
        '                                     Name="NodeTemplatePath"/>',
        '                            <Image   Grid.Column="2"',
        '                                     Name="NodeTemplatePathIcon"/>',
        '                            <Button  Grid.Column="3"',
        '                                     Name="NodeTemplatePathBrowse" ',
        '                                     Content="Browse"/>',
        '                        </Grid>',
        '                    </Grid>',
        '                </Grid>',
        '            </Grid>',
        '        </TabItem>',
        '    </TabControl>',
        '</Window>' -join "`n")
    }
```

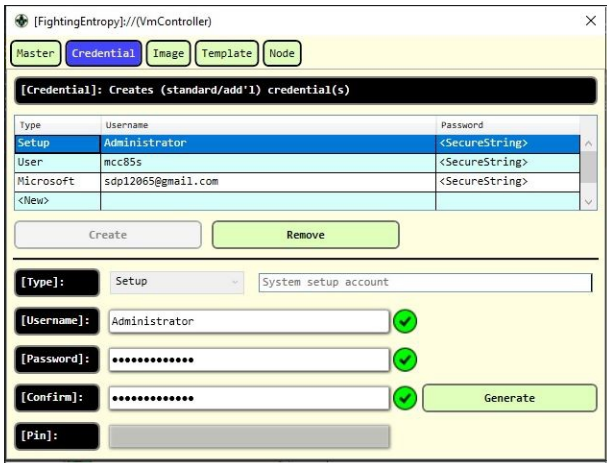


[Master]: Propagates valid template properties

| Status | Alias | Description |
|---|---|---|
| Up | vEthernet (External) | Hyper-V Virtual Ethernet Adapter #2 |

| | | | |
|---|---|---|---|
| [Path]: | C:\FileVm | | Browse |
| [Domain]: | securedigitsplus.com | [NetBios]: secured | Create |

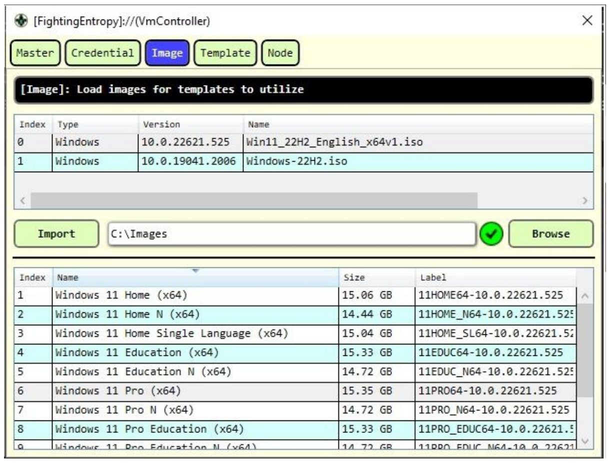
| Name | Value |
|---|---|
| ComputerName | L420-X64 |
| Alias | vEthernet (External) |
| Description | Hyper-V Virtual Ethernet Adapter #2 |
| CompID | 1 |
| CompDescription | Default Compartment |
| MacAddress | DE-56-7D-78-94-3A |
| Status | Up |



[Credential]: Creates (standard/add'l) credential(s)

| Type | Username | Password |
|---|---|---|
| Setup | Administrator | <SecureString> |
| User | mcc85s | <SecureString> |
| Microsoft | sdp12065@gmail.com | <SecureString> |
| <New> | | |

| | | |
|---|---|---|
| [Type]: | Setup | System setup account |
| [Username]: | Administrator | |
| [Password]: | ••••••••••••• | |
| [Confirm]: | ••••••••••••• | Generate |
| [Pin]: | | |

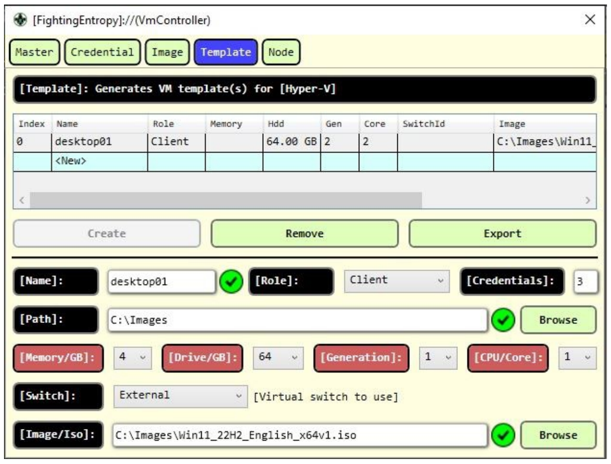


[Image]: Load images for templates to utilize

| Index | Type | Version | Name |
|---|---|---|---|
| 0 | Windows | 10.0.22621.525 | Win11_22H2_English_x64v1.iso |
| 1 | Windows | 10.0.19041.2006 | Windows-22H2.iso |

C:\Images

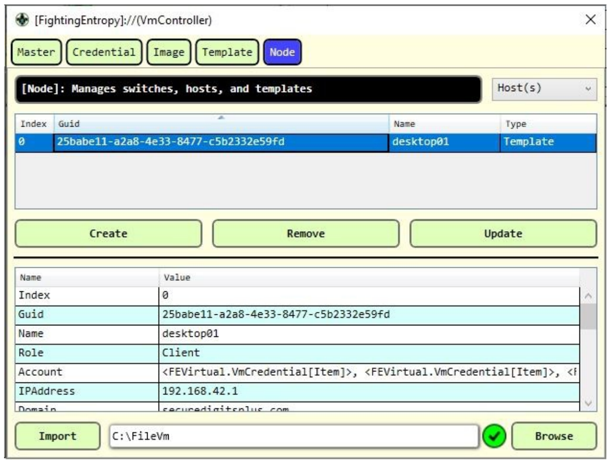
| Index | Name | Size | Label |
|---|---|---|---|
| 1 | Windows 11 Home (x64) | 15.06 GB | 11HOME64-10.0.22621.525 |
| 2 | Windows 11 Home N (x64) | 14.44 GB | 11HOME_N64-10.0.22621.525 |
| 3 | Windows 11 Home Single Language (x64) | 15.04 GB | 11HOME_SL64-10.0.22621.52 |
| 4 | Windows 11 Education (x64) | 15.33 GB | 11EDUC64-10.0.22621.525 |
| 5 | Windows 11 Education N (x64) | 14.72 GB | 11EDUC_N64-10.0.22621.525 |
| 6 | Windows 11 Pro (x64) | 15.35 GB | 11PRO64-10.0.22621.525 |
| 7 | Windows 11 Pro N (x64) | 14.72 GB | 11PRO_N64-10.0.22621.525 |
| 8 | Windows 11 Pro Education (x64) | 15.33 GB | 11PRO_EDUC64-10.0.22621.5 |
| 9 | Windows 11 Pro Education N (x64) | 14.72 GB | 11PRO_EDUC_N64-10.0.22621 |



[Template]: Generates VM template(s) for [Hyper-V]

| Index | Name | Role | Memory | Hdd | Gen | Core | SwitchId | Image |
|---|---|---|---|---|---|---|---|---|
| 0 | desktop01 | Client | | 64.00 GB | 2 | 2 | | C:\Images\Win11_ |
| | <New> | | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| [Name]: | desktop01 | [Role]: | Client | [Credentials]: | 3 | | |
| [Path]: | C:\Images | | | | | | Browse |
| [Memory/GB]: | 4 | [Drive/GB]: | 64 | [Generation]: | 1 | [CPU/Core]: | 1 |
| [Switch]: | External | [Virtual switch to use] | | | | | |
| [Image/Iso]: | C:\Images\Win11_22H2_English_x64v1.iso | | | | | | Browse |



[Node]: Manages switches, hosts, and templates
Host(s)

| Index | Guid | Name | Type |
|---|---|---|---|
| 0 | 25babe11-a2a8-4e33-8477-c5b2332e59fd | desktop01 | Template |

| Name | Value |
|---|---|
| Index | 0 |
| Guid | 25babe11-a2a8-4e33-8477-c5b2332e59fd |
| Name | desktop01 |
| Role | Client |
| Account | <FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>, <F |
| IPAddress | 192.168.42.1 |
| Domain | securedigitsplus.com |

C:\FileVm

```
                                                                                            ____________________________/
\__________________________________________________________________________________________/ Class [VmControllerXaml]
  Class [VmMain] /----------------------------------------------------------------------------------------------------------\
/----------------
```

    Specifically meant to contain information from the [Main panel], for:
    [+] VM template path
    [+] Domain name
    [+] NetBios ID

```
Class VmMain
{
    [String]    $Path
    [String]  $Domain
    [String] $NetBios
    VmMain([String]$Path,[String]$Domain,[String]$NetBios)
    {
        $This.Path    = $Path
        $This.Domain  = $Domain.ToLower()
        $This.NetBios = $NetBios.ToUpper()
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmMain>"
    }
}
```

All of these fields are propogated into the [VmTemplate] and [VmNode] objects.

```
PS Prompt:\> $Ctrl.Master.Main

Path      Domain               NetBios
----      ------               -------
C:\FileVm securedigitsplus.com SECURED

PS Prompt:\>
```

```
                                                                                                                   ________________/
\___________________________________________________________________________________________________________/ Class [VmMain]
  Class [VmNetworkConfig] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

This is ALMOST a verbatim copy of the object returned from [Get-NetIpConfiguration -Detailed], for each config. However, some of the fields and properties have been altered so that it is more flattened than spread out as a complicated object. This helps to extract information needed to templatize the [VmTemplate] objects. This object is seen in the first sub tab item [Config] under first main tab item [Master].

```
Class VmNetworkConfig
{
    Hidden [Object]        $Config
    [String]         $ComputerName
    [String]                $Alias
    [String]          $Description
    [String]               $CompID
    [String]      $CompDescription
    [String]           $MacAddress
    [String]               $Status
    [String]                 $Name
    [String]             $Category
    [String]     $IPv4Connectivity
    [String]          $IPv4Address
    [String]           $IPv4Prefix
    [String]   $IPv4DefaultGateway
    [String]     $IPv4InterfaceMtu
    [String]    $IPv4InterfaceDhcp
    [String[]]       $IPv4DnsServer
    [String]     $IPv6Connectivity
    [String] $IPv6LinkLocalAddress
    [String]   $IPv6DefaultGateway
    [String]     $IPv6InterfaceMtu
    [String]    $IPv6InterfaceDhcp
    [String[]]       $IPv6DnsServer
    VmNetworkConfig([Object]$Config)
    {
        $This.Config                 = $Config
        $This.ComputerName           = $Config.ComputerName
        $This.Alias                  = $Config.InterfaceAlias
        $This.Description            = $Config.InterfaceDescription
```

```
            $This.CompID                  = $Config.NetCompartment.CompartmentId
            $This.CompDescription         = $Config.NetCompartment.CompartmentDescription
            $This.MacAddress              = $Config.NetAdapter.LinkLayerAddress
            $This.Status                  = $Config.NetAdapter.Status
            $This.Name                    = $Config.NetProfile.Name
            $This.Category                = $Config.NetProfile.NetworkCategory
            $This.IPv4Connectivity        = $Config.NetProfile.IPv4Connectivity
            $This.IPv4Address             = $Config.IPv4Address.IpAddress
            $This.IPv4Prefix              = $Config.IPv4Address.PrefixLength
            $This.IPv4DefaultGateway      = $Config.IPv4DefaultGateway.NextHop
            $This.IPv4InterfaceMtu        = $Config.NetIPv4Interface.NlMTU
            $This.IPv4InterfaceDhcp       = $Config.NetIPv4Interface.DHCP
            $This.IPv4DnsServer           = $Config.DNSServer | ? AddressFamily -eq 2 | % ServerAddresses
            $This.IPv6Connectivity        = $Config.NetProfile.IPv6Connectivity
            $This.IPv6DefaultGateway      = $Config.IPv6DefaultGateway.NextHop
            $This.IPv6LinkLocalAddress    = $Config.IPv6LinkLocalAddress
            $This.IPv6InterfaceMtu        = $Config.NetIPv6Interface.NlMTU
            $This.IPv6InterfaceDhcp       = $Config.NetIPv6Interface.DHCP
            $This.IPv6DnsServer           = $Config.DNSServer | ? AddressFamily -eq 23 | % ServerAddresses
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNetwork[Config]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Master.Config

ComputerName         : L420-X64
Alias                : vEthernet (External)
Description          : Hyper-V Virtual Ethernet Adapter #2
CompID               : 1
CompDescription      : Default Compartment
MacAddress           : DE-56-7D-78-94-3A
Status               : Up
Name                 : Network 373
Category             : Public
IPv4Connectivity     : Internet
IPv4Address          : 192.168.42.2
IPv4Prefix           : 24
IPv4DefaultGateway   : 192.168.42.129
IPv4InterfaceMtu     : 1500
IPv4InterfaceDhcp    : Enabled
IPv4DnsServer        : {192.168.42.129}
IPv6Connectivity     : NoTraffic
IPv6LinkLocalAddress : fe80::bf96:b672:7015:7147%26
IPv6DefaultGateway   :
IPv6InterfaceMtu     : 1500
IPv6InterfaceDhcp    : Enabled
IPv6DnsServer        :

PS Prompt:\>
```

```
                                                                                                            ________________________/
\__________________________________________________________________________________________/ Class [VmNetworkConfig]
  Class [VmNetworkHost] /-----------------------------------------------------------------------------------------------\
/----------------------
```

Object returned from a ping sweep on a given network. The host object contains [IPAddress], [Hostname], [Alias], and [AddressList] fields for [System.Net.Dns] to resolve at a later point if a node is found.

```
    Class VmNetworkHost
    {
        [UInt32]         $Index
        [UInt32]        $Status
        [String]          $Type = "Host"
        [String]     $IpAddress
        [String]      $Hostname
        [String[]]     $Aliases
        [String[]] $AddressList
        VmNetworkHost([UInt32]$Index,[String]$IpAddress,[Object]$Reply)
        {
            $This.Index          = $Index
            $This.Status         = $Reply.Result.Status -match "Success"
            $This.IpAddress      = $IpAddress
        }
        VmNetworkHost([UInt32]$Index,[String]$IpAddress)
        {
            $This.Index          = $Index
            $This.Status         = 0
            $This.IpAddress      = $IpAddress
        }
        Resolve()
        {
            $Item                = [System.Net.Dns]::Resolve($This.IpAddress)
            $This.Hostname       = $Item.Hostname
            $This.Aliases        = $Item.Aliases
            $This.AddressList    = $Item.AddressList
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNetwork[Host]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Master.Network.Hosts[2] | Format-List

Index       : 2
Status      : 1
Type        : Host
IpAddress   : 192.168.42.2
Hostname    : l420-x64.securedigitsplus.com
Aliases     : {}
AddressList : {192.168.42.2}

PS Prompt:\>
```

```
                                                                                                   ______________________/
\_____________________________________________________________________________________________/ Class [VmNetworkHost]
  Class [VmNetworkBase] /---------------------------------------------------------------------------------------------\
/-----------------------
```

Contains all of the information needed to realize the entire host range as well as [DHCP] settings for forward [DHCP/DNS/ADDS/WDS/IIS/MDT] servers to utilize.

```
    Class VmNetworkBase
    {
        [String]    $Domain
        [String]   $NetBios
        [String]   $Network
        [String] $Broadcast
        [String]   $Trusted
        [UInt32]    $Prefix
        [String]   $Netmask
        [String]  $Wildcard
        [String]   $Gateway
        [String[]]     $Dns
        VmNetworkBase([Object]$Main,[Object]$Config)
        {
            $This.Domain     = $Main.Domain
            $This.NetBios    = $Main.NetBios
            $This.Trusted    = $Config.IPV4Address
            $This.Prefix     = $Config.IPv4Prefix

            # Binary
            $This.GetConversion()

            $This.Gateway    = $Config.IPV4DefaultGateway
            $This.Dns       = $Config.IPv4DnsServer
        }
        GetConversion()
        {
            # Convert IP and PrefixLength into binary, netmask, and wildcard
            $xBinary       = 0..3 | % { (($_*8)..(($_*8)+7) | % { @(0,1)[$_ -lt $This.Prefix] }) -join '' }
            $This.Netmask  = ($xBinary | % { [Convert]::ToInt32($_,2 ) }) -join "."
            $This.Wildcard = ($This.Netmask.Split(".") | % { (256-$_) }) -join "."
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNetwork[Base]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Master.Network.Base

Domain    : securedigitsplus.com
NetBios   : SECURED
Network   : 192.168.42.0
Broadcast : 192.168.42.255
Trusted   : 192.168.42.2
Prefix    : 24
Netmask   : 255.255.255.0
Wildcard  : 1.1.1.256
Gateway   : 192.168.42.129
Dns       : {192.168.42.129}

PS Prompt:\>
```

```
                                                                                                              ______________________/
\____________________________________________________________________________________________________________/ Class [VmNetworkBase]
  Class [VmNetworkDhcp] /----------------------------------------------------------------------------------------------\
/-----------------------
```

Used to forward [Dhcp] information to [nodes] and [servers].

```

    Class VmNetworkDhcp
    {
        [String]          $Name
        [String]    $SubnetMask
        [String]       $Network
        [String]    $StartRange
        [String]      $EndRange
        [String]     $Broadcast
        [String[]]   $Exclusion
        VmNetworkDhcp([Object]$Base,[Object]$Hosts)
        {
            $This.Network     = $Base.Network   = $Hosts[0].IpAddress
            $This.Broadcast   = $Base.Broadcast = $Hosts[-1].IpAddress
            $This.Name        = "{0}/{1}" -f $This.Network, $Base.Prefix
            $This.SubnetMask  = $Base.Netmask
            $Range            = $Hosts | ? Type -eq Host
            $This.StartRange  = $Range[0].IpAddress
            $This.EndRange    = $Range[-1].IpAddress
            $This.Exclusion   = $Range | ? Status | % IpAddress
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNetwork[Dhcp]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Master.Network.Dhcp

Name       : 192.168.42.0/24
```

```
SubnetMask : 255.255.255.0
Network    : 192.168.42.0
StartRange : 192.168.42.1
EndRange   : 192.168.42.254
Broadcast  : 192.168.42.255
Exclusion  : {192.168.42.2, 192.168.42.129}

PS Prompt:\>
```

```
                                                                                                             ______________________/
\___________________________________________________________________________________________________________/ Class [VmNetworkDhcp]
  Class [VmNetworkNode] /----------------------------------------------------------------------------------------------------------\
/-----------------------
```

This object is meant specifically for [combining information] into a [single node] for the [template] to propagate the [correct information] to the node upon [instantiation + realization] of the [template file].

```
Class VmNetworkNode
{
    [UInt32]     $Index
    [String]      $Name
    [String] $IpAddress
    [String]    $Domain
    [String]   $NetBios
    [String]   $Trusted
    [UInt32]    $Prefix
    [String]   $Netmask
    [String]   $Gateway
    [String[]]     $Dns
    [Object]      $Dhcp
    VmNetworkNode([UInt32]$Index,[String]$Name,[String]$IpAddress,[Object]$Hive)
    {
        $This.Index     = $Index
        $This.Name      = $Name
        $This.IpAddress = $IpAddress
        $This.Domain    = $Hive.Domain
        $This.NetBios   = $Hive.NetBios
        $This.Trusted   = $Hive.Trusted
        $This.Prefix    = $Hive.Prefix
        $This.Netmask   = $Hive.Netmask
        $This.Gateway   = $Hive.Gateway
        $This.Dns       = $Hive.Dns
        $This.Dhcp      = $Hive.Dhcp
    }
    VmNetworkNode([Object]$File)
    {
        $This.Index     = $File.Index
        $This.Name      = $File.Name
        $This.IpAddress = $File.IpAddress
        $This.Domain    = $File.Domain
        $This.NetBios   = $File.NetBios
        $This.Trusted   = $File.Trusted
        $This.Prefix    = $File.Prefix
        $This.Netmask   = $File.Netmask
        $This.Gateway   = $File.Gateway
        $This.Dns       = $File.Dns
        $This.Dhcp      = $File.Dhcp
    }
    [String] Hostname()
    {
        Return "{0}.{1}" -f $This.Name, $This.Domain
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNetwork[Node]>"
    }
}
```

```
PS Prompt:\> $Ctrl.Template.VmTemplateNetwork($Ctrl.Master.Network)

IpAddress : 192.168.42.1
Domain    : securedigitsplus.com
NetBios   : SECURED
Trusted   : 192.168.42.2
Prefix    : 24
Netmask   : 255.255.255.0
Gateway   : 192.168.42.129
Dns       : {192.168.42.129}
Dhcp      : <FEVirtual.VmNetwork[Dhcp]>

PS Prompt:\>
```

```
                                                                                                ______________________/
\______________________________________________________________________________________________/ Class [VmNetworkNode]
  Class [VmNetworkRange] /---------------------------------------------------------------------------------------------\
/------------------------
```

```
Contains information about a [single network range] within a possible array[] of other subnetworks.

Based on the [netmask] and the [notation string], it expands the [notation] into an array[] of IP addresses so
that they can be [scanned] using the [ping sweep] in the [VmControllerMaster] object.
```

```
    Class VmNetworkRange
    {
        [UInt32]     $Index
        [String]     $Count
        [String]   $Netmask
        [String]  $Notation
        [Object]    $Output
        VmNetworkRange([UInt32]$Index,[String]$Netmask,[UInt32]$Count,[String]$Notation)
        {
            $This.Index    = $Index
            $This.Count    = $Count
            $This.Netmask  = $Netmask
            $This.Notation = $Notation
            $This.Output   = @( )
        }
        Expand()
        {
            $Split     = $This.Notation.Split("/")
            $HostRange = @{ }
            ForEach ($0 in $Split[0] | Invoke-Expression)
            {
                ForEach ($1 in $Split[1] | Invoke-Expression)
                {
                    ForEach ($2 in $Split[2] | Invoke-Expression)
                    {
                        ForEach ($3 in $Split[3] | Invoke-Expression)
                        {
                            $HostRange.Add($HostRange.Count,"$0.$1.$2.$3")
                        }
                    }
                }
            }

            $This.Output    = $HostRange[0..($HostRange.Count-1)]
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNetwork[Range]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Master.Network.Range

Index    : 0
```

```
Count    : 256
Netmask  : 255.255.255.0
Notation : 192/168/42/0..255
Output   : {192.168.42.0, 192.168.42.1, 192.168.42.2, 192.168.42.3...}

PS Prompt:\>
```

```
                                                                                                              ________________________/
\_____________________________________________________________________________________________________________/ Class [VmNetworkRange]
  Class [VmNetworkControl] /---------------------------------------------------------------------------------------------\
/--------------------------
```

Combines all of the above [network classes] that have a [property] of the [same name], below.

Also, this class searches for the [subnetwork] that has the [current IP address] from the selected [configuration], and it [expands that network] to search for [possible hostnames].

```
Class VmNetworkControl
{
    [Object]   $Config
    [Object]     $Base
    [Object]    $Range
    [Object]    $Hosts
    [Object]     $Dhcp
    VmNetworkControl([Object]$Main,[Object]$Config)
    {
        $This.Config   = $Config
        $This.Base     = $This.VmNetworkBase($Main,$Config)
        $This.Range    = @( )
        $This.Hosts    = @( )

        $This.GetNetworkRange()
    }
    [Object] VmNetworkBase([Object]$Main,[Object]$Config)
    {
        Return [VmNetworkBase]::New($Main,$Config)
    }
    [Object] VmNetworkRange([UInt32]$Index,[String]$Netmask,[UInt32]$Count,[String]$Notation)
    {
        Return [VmNetworkRange]::New($Index,$Netmask,$Count,$Notation)
    }
    [Object] VmNetworkDhcp([Object]$Base,[Object[]]$Hosts)
    {
        Return [VmNetworkDhcp]::New($Base,$Hosts)
    }
    [Object] VmNetworkHost([UInt32]$Index,[String]$IpAddress)
    {
        Return [VmNetworkHost]::New($Index,$IpAddress)
    }
    AddList([UInt32]$Count,[String]$Notation)
    {
        $This.Range += $This.VmNetworkRange($This.Range.Count,$This.Base.Netmask,$Count,$Notation)
    }
    GetNetworkRange()
    {
        $Address       = $This.Base.Trusted.Split(".")

        $xNetmask      = $This.Base.Netmask  -split "\."
        $xWildCard     = $This.Base.Wildcard -split "\."
        $Total         = $xWildcard -join "*" | Invoke-Expression

        # Convert wildcard into total host range
        $Hash          = @{ }
        ForEach ($X in 0..3)
        {
            $Value = Switch ($xWildcard[$X])
            {
                1
                {
                    $Address[$X]
                }
                Default
                {
                    ForEach ($Item in 0..255 | ? { $_ % $xWildcard[$X] -eq 0 })
                    {
                        "{0}..{1}" -f $Item, ($Item+($xWildcard[$X]-1))
                    }
                }
                255
                {
                    "{0}..{1}" -f $xNetmask[$X],($xNetmask[$X]+$xWildcard[$X])
                }
            }

            $Hash.Add($X,$Value)
        }

        # Build host range
        $xRange   = @{ }
        ForEach ($0 in $Hash[0])
        {
            ForEach ($1 in $Hash[1])
            {
                ForEach ($2 in $Hash[2])
                {
                    ForEach ($3 in $Hash[3])
                    {
                        $xRange.Add($xRange.Count,"$0/$1/$2/$3")
                    }
                }
            }
        }

        Switch ($xRange.Count)
        {
            0
            {
                "Error"
            }
            1
            {
                $This.AddList($Total,$xRange[0])
            }
            Default
            {
                ForEach ($X in 0..($xRange.Count-1))
                {
                    $This.AddList($Total,$xRange[$X])
                }
            }
        }

        # Subtract network + broadcast addresses
        ForEach ($Network in $This.Range)
        {
            $Network.Expand()
            If ($This.Base.Trusted -in $Network.Output)
            {
                $xHost               = @{ }
                ForEach ($Item in $Network.Output)
                {
                    $xHost.Add($xHost.Count,$This.VmNetworkHost($xHost.Count,$Item))
                }
                $This.Hosts          = $xHost[0..($xHost.Count-1)]
                $This.Hosts[ 0].Type = "Network"
                $This.Hosts[-1].Type = "Broadcast"
            }
            Else
            {
                $Network.Output      = @( )
            }
        }
```

```
        }
        SetDhcp()
        {
            $This.Dhcp      = $This.VmNetworkDhcp($This.Base,$This.Hosts)
        }
        [String] FirstAvailableIPAddress()
        {
            $Address = $Null
            $List    = $This.Hosts | ? Type -eq Host | ? Status -eq 0
            If ($List.Count -gt 0)
            {
                $Address = $List[0].IPAddress
            }

            Return $Address
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNetwork[Control]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Master.Network.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    VmNetworkControl                         System.Object

PS Prompt:\> $Ctrl.Master.Network

Config : <FEVirtual.VmNetwork[Config]>
Base   : <FEVirtual.VmNetwork[Base]>
Range  : {<FEVirtual.VmNetwork[Range]>}
Hosts  : {<FEVirtual.VmNetwork[Host]>, <FEVirtual.VmNetwork[Host]>, <FEVirtual.VmNetwork[Host]>,
         <FEVirtual.VmNetwork[Host]>...}
Dhcp   : <FEVirtual.VmNetwork[Dhcp]>

PS Prompt:\>
```

```
                                                                                                            ___________________________/
\___________________________________________________________________________________________________________/ Class [VmNetworkControl]
  Class [VmNetworkMaster] /---------------------------------------------------------------------------------------------\
/------------------------
```

This class [combines] a number of the [above classes] so that it can [orchestrate] the [templatization] of an [available IP address], with the [associated network], [maximum host count], [possible available hosts], [DHCP options], et cetera.

Effectively, this class combines all of these properties so that it knows whether to [throw an error] if a [duplicate name] or [host] is found on the [network], whereby [emulating an actual DNS server].

```

    Class VmNetworkMaster
    {
        [Object]       $Main
        [Object]      $Config
        [Object]     $Network
        VmNetworkMaster()
        {
            $This.Config = $This.VmNetworkConfig()
        }
        [Object[]] NetIPConfig()
        {
            Return Get-NetIPConfiguration -Detailed | ? IPV4DefaultGateway
        }
        [Object] VmMain([String]$Path,[String]$Domain,[String]$NetBios)
        {
            Return [VmMain]::New($Path,$Domain,$NetBios)
        }
        [Object[]] VmNetworkConfig()
        {
            Return $This.NetIPConfig() | % { [VmNetworkConfig]::New($_) }
        }
        [Object] VmNetworkControl([Object]$Main,[Object]$Config)
        {
            Return [VmNetworkControl]::New($Main,$Config)
        }
        SetMain([String]$Path,[String]$Domain,[String]$NetBios)
        {
            $This.Main = $This.VmMain($Path,$Domain,$NetBios)
        }
        SetNetwork([UInt32]$Index)
        {
            If (!$This.Main)
            {
                Throw "Must set (Path/Domain/NetBios) info first"
            }

            ElseIf ($Index -gt $This.Config.Count)
            {
                Throw "Invalid index"
            }

            $This.Network = $This.VmNetworkControl($This.Main,$This.Config[$Index])
        }
        InternalPingSweep()
        {
            If ($This.Network.Range.Output.Count -eq 0)
            {
                Throw "Unable to run the scan"
            }

            $xHosts   = $This.Network.Hosts.IPAddress
            $Buffer   = 97..119 + 97..105 | % { "0x{0:X}" -f $_ }
            $Option   = New-Object System.Net.NetworkInformation.PingOptions
            $Ping     = @{ }
            ForEach ($X in 0..($xHosts.Count-1))
            {
                $Item = New-Object System.Net.NetworkInformation.Ping
                $Ping.Add($X,$Item.SendPingAsync($xHosts[$X],100,$Buffer,$Option))
            }

            ForEach ($X in 0..($Ping.Count-1))
            {
                $This.Network.Hosts[$X].Status = [UInt32]($Ping[$X].Result.Status -eq "Success")
            }
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNetwork[Master]>"
        }
    }
```

```
                                                                                                         _________________________/
\_______________________________________________________________________________________________________/ Class [VmNetworkMaster]
  Class [VmCredentialType] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Meant strictly for the GUI, these are the types that I selected for propagating [actual accounts].

```
    Enum VmCredentialType
    {
        Setup
        System
        Service
        User
        Microsoft
    }
```

```
                                                                                                 ________________________/
\_______________________________________________________________________________________________/ Class [VmCredentialType]
 Class [VmCredentialSlot] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

This is meant to provide an [Object] for the [enum type] to slot itself into, whereby obtaining a [numerical index] and a [description].

```
Class VmCredentialSlot
{
    [UInt32]       $Index
    [String]        $Name
    [String] $Description
    VmCredentialSlot([String]$Name)
    {
        $This.Index = [UInt32][VmCredentialType]::$Name
        $This.Name  = [VmCredentialType]::$Name
    }
    [String] ToString()
    {
        Return $This.Name
    }
}
```

```
PS Prompt:\> $Ctrl.Credential.Slot[0] | Format-List

Index       : 0
Name        : Setup
Description : System setup account

PS Prompt:\>
```

```
                                                                                                 ________________________/
\_______________________________________________________________________________________________/ Class [VmCredentialSlot]
 Class [VmCredentialList] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

This combines the [enum types], and [slots] into an [Object] that can be controlled by the [GUI ComboBox].

```
Class VmCredentialList
{
    [Object] $Output
    VmCredentialList()
    {
        $This.Refresh()
    }
    [Object] VmCredentialSlot([String]$Name)
    {
        Return [VmCredentialSlot]::New($Name)
    }
    Clear()
    {
        $This.Output = @( )
    }
    Refresh()
    {
        $This.Clear()

        ForEach ($Name in [System.Enum]::GetNames([VmCredentialType]))
        {
            $Item             = $This.VmCredentialSlot($Name)
            $Item.Description = Switch ($Item.Name)
```

```
                {
                    Setup     { "System setup account"       }
                    System    { "System level account"       }
                    Service   { "Service level account"      }
                    User      { "Local/domain user account"  }
                    Microsoft { "Online Microsoft account"   }
                }

                $This.Add($Item)
            }
        }
        Add([Object]$Object)
        {
            $This.Output += $Object
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmCredential[Type[]]"
        }
    }
```

```
PS Prompt:\> [VmCredentialList]::New()

Output
------
{Setup, System, Service, User...}

PS Prompt:\>
```

```
                                                                                                          ___________________________/
\________________________________________________________________________________________________/ Class [VmCredentialList]
  Class [VmCredentialItem] /--------------------------------------------------------------------------------------------\
/--------------------------
```

With the above [credential slot list], it is possible to create individual objects that adhere to the conventions of each [individual account] or [credential type]. This is to [emulate credentials] or to [create user objects] for [Active Directory], or a range of other [applications]. It is also meant to correctly deserialize the template.

```
    Class VmCredentialItem
    {
        [UInt32]            $Index
        [Guid]               $Guid
        [Object]             $Type
        [String]         $Username
        Hidden [String]      $Pass
        [PSCredential] $Credential
        [String]              $Pin
        VmCredentialItem([UInt32]$Index,[Object]$Type,[PSCredential]$Credential)
        {
            $This.Index      = $Index
            $This.Guid       = $This.NewGuid()
            $This.Type       = $Type
            $This.Username   = $Credential.Username
            $This.Credential = $Credential
            $This.Pass       = $This.Mask()
        }
        VmCredentialItem([Object]$Serial)
        {
            $This.Index      = $Serial.Index
            $This.Guid       = $Serial.Guid
            $This.Type       = $Serial.Type
            $This.Username   = $Serial.Username
            $This.Credential = $Serial.Credential
            $This.Pass       = $This.Mask()
            $This.Pin        = $Serial.Pin
        }
        [Object] NewGuid()
        {
            Return [Guid]::NewGuid()
        }
        [String] Password()
        {
            Return $This.Credential.GetNetworkCredential().Password
        }
        [String] Mask()
        {
            Return "<SecureString>"
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmCredential[Item]>"
        }
    }
```

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

Index      : 0
Guid       : 0c98ed6c-7a92-4dd4-bb05-b648387984f2
Type       : Setup
Username   : Administrator
Credential : System.Management.Automation.PSCredential
Pin        :

PS Prompt:\>
```

```
                                                                                                             ___________________________/
\___________________________________________________________________________________________________________/ Class [VmCredentialItem]
  Class [VmCredentialMaster] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
    This is basically a controller meant to control the above [credential classes], as well as the GUI.
    It provides groundwork for [validation] and can be [extended] to throw more [restrictions], [conventions],
    [rules], or even [properties] as needed...

    For instance, a [Personal Identification Number] for a [Microsoft account].
```

```
    Class VmCredentialMaster
    {
        [String]        $Name
        Hidden [Object] $Slot
        [UInt32]       $Count
        [Object]      $Output
        VmCredentialMaster()
        {
            $This.Name = "VmCredentialMaster"
            $This.Slot = $This.VmCredentialList()
            $This.Clear()
        }
        Clear()
        {
            $This.Output = @( )
            $This.Count  = 0
            $This.Setup()
        }
        [Object] VmCredentialList()
        {
            Return [VmCredentialList]::New().Output
        }
        [Object] VmCredentialItem([UInt32]$Index,[String]$Type,[PSCredential]$Credential)
        {
            Return [VmCredentialItem]::New($Index,$Type,$Credential)
        }
        [Object] VmCredentialItem([Object]$Serial)
        {
            Return [VmCredentialItem]::New($Serial)
        }
        [PSCredential] SetCredential([String]$Username,[String]$Pass)
        {
            Return [PSCredential]::New($Username,$This.SecureString($Pass))
        }
        [PSCredential] SetCredential([String]$Username,[SecureString]$Pass)
        {
            Return [PSCredential]::New($Username,$Pass)
        }
        [SecureString] SecureString([String]$In)
        {
            Return $In | ConvertTo-SecureString -AsPlainText -Force
        }
        [String] Generate()
        {
            Do
            {
                $Length          = $This.Random(10,16)
                $Bytes           = [Byte[]]::New($Length)

                ForEach ($X in 0..($Length-1))
                {
                    $Bytes[$X]   = $This.Random(32,126)
                }

                $Pass            = [Char[]]$Bytes -join ''
            }
            Until ($Pass -match $This.Pattern())

            Return $Pass
        }
        [String] Pattern()
        {
            Return "(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[:punct:]).{10}"
        }
        [UInt32] Random([UInt32]$Min,[UInt32]$Max)
        {
            Return Get-Random -Min $Min -Max $Max
        }
        Setup()
        {
            If ("Administrator" -in $This.Output.Username)
            {
                Throw "Administrator account already exists"
            }

            $This.Add(0,"Administrator",$This.Generate())
        }
        Rerank()
        {
            $C = 0
            ForEach ($Item in $This.Output)
            {
                $Item.Index = $C
                $C ++
            }
        }
        Add([UInt32]$Type,[String]$Username,[String]$Pass)
        {
            If ($Type -gt $This.Slot.Count)
            {
                Throw "Invalid account type"
            }

            $Credential   = $This.SetCredential($Username,$Pass)
            $This.Output += $This.VmCredentialItem($This.Count,$This.Slot[$Type],$Credential)
            $This.Count   = $This.Output.Count
        }
        Add([UInt32]$Type,[String]$Username,[SecureString]$Pass)
        {
            If ($Type -gt $This.Slot.Count)
            {
                Throw "Invalid account type"
            }
            $Credential    = $This.SetCredential($Username,$Pass)
            $This.Output += $This.VmCredentialItem($This.Count,$This.Slot[$Type],$Credential)
            $This.Count    = $This.Output.Count
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmCredential[Master]"
        }
    }
```

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

Name               Count Output
----               ----- ------
VmCredentialMaster     3 {<FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>, <FEVirtual.VmCrede...

PS Prompt:\>
```

```
                                                                                                   ____________________________/
\__________________________________________________________________________________________/ Class [VmCredentialMaster]
  Class [VmByteSize] /---------------------------------------------------------------------------------------------------\
/--------------------
```

*Virtually* the same object as [ImageByteSize]
This class is essentially meant to make a byte size [UInt64] more consumable to look at.

```
Class VmByteSize
{
    [String]   $Name
    [UInt64]  $Bytes
    [String]   $Unit
    [String]   $Size
    VmByteSize([String]$Name,[UInt64]$Bytes)
    {
        $This.Name   = $Name
        $This.Bytes  = $Bytes
        $This.GetUnit()
        $This.GetSize()
    }
    GetUnit()
    {
        $This.Unit   = Switch ($This.Bytes)
        {
            {$_ -lt 1KB}                 {     "Byte" }
            {$_ -ge 1KB -and $_ -lt 1MB} { "Kilobyte" }
            {$_ -ge 1MB -and $_ -lt 1GB} { "Megabyte" }
            {$_ -ge 1GB -and $_ -lt 1TB} { "Gigabyte" }
            {$_ -ge 1TB}                 { "Terabyte" }
        }
    }
    GetSize()
    {
        $This.Size   = Switch -Regex ($This.Unit)
        {
            ^Byte     {     "{0} B" -f  $This.Bytes/1    }
            ^Kilobyte { "{0:n2} KB" -f ($This.Bytes/1KB) }
            ^Megabyte { "{0:n2} MB" -f ($This.Bytes/1MB) }
            ^Gigabyte { "{0:n2} GB" -f ($This.Bytes/1GB) }
            ^Terabyte { "{0:n2} TB" -f ($This.Bytes/1TB) }
        }
    }
    [String] ToString()
    {
        Return $This.Size
    }
}
```

```
                                                                                                     ____________________/
\___________________________________________________________________________________________________/ Class [VmByteSize]
  Class [VmRole] /--------------------------------------------------------------------------------------------------------\
/----------------
```

Meant to further define the [type of template], to be expanded at a later time.

```
    Class VmRole
    {
        [UInt32]  $Index
        [String]   $Type
        VmRole([UInt32]$Index)
        {
            $This.Index = $Index
            $This.Type  = @("Server","Client","Unix")[$Index]
        }
        [String] ToString()
        {
            Return $This.Type
        }
    }
```

```
PS Prompt:\> $Ctrl.Template.Output[0].Role.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    VmRole                                   System.Object

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

Index Type
----- ----
    1 Client

PS Prompt:\>
```

```
                                                                                                         ________________/
\_______________________________________________________________________________________________________/ Class [VmRole]
  Class [VmTemplateNetwork] /---------------------------------------------------------------------------------------------\
/----------------------------
```

Meant to compartmentalize the necessary information for the [template file].

It also pulls the [first available IP address] so that it [does not conflict with other hosts on the network].

```
    Class VmTemplateNetwork
    {
        [String] $IpAddress
        [String]    $Domain
        [String]   $NetBios
        [String]   $Trusted
        [UInt32]    $Prefix
        [String]   $Netmask
        [String]   $Gateway
        [String[]]     $Dns
        [Object]      $Dhcp
        VmTemplateNetwork([Object]$Network)
        {
            $This.IPAddress = $Network.FirstAvailableIPAddress()
            $This.Domain    = $Network.Base.Domain
            $This.NetBios   = $Network.Base.NetBios
            $This.Trusted   = $Network.Base.Trusted
            $This.Prefix    = $Network.Base.Prefix
            $This.Netmask   = $Network.Base.Netmask
            $This.Gateway   = $Network.Base.Gateway
            $This.Dns       = $Network.Base.Dns
            $This.Dhcp       = $Network.Dhcp
        }
    }
```

```
PS Prompt:\> $Ctrl.Template.VmTemplateNetwork($Ctrl.Master.Network)

IpAddress : 192.168.42.1
Domain    : securedigitsplus.com
NetBios   : SECURED
Trusted   : 192.168.42.2
Prefix    : 24
Netmask   : 255.255.255.0
Gateway   : 192.168.42.129
Dns       : {192.168.42.129}
Dhcp      : <FEVirtual.VmNetwork[Dhcp]>

PS Prompt:\>
```

```
                                                                                            ___________________________/
\___________________________________________________________________________________________/ Class [VmTemplateNetwork]
  Class [VmTemplateItem] /---------------------------------------------------------------------------------------------\
/------------------------
```

```
Combines the [networking information] and the [selected properties in the GUI] into a template.
Though the same can also be done from the [PowerShell CLI] by using the same method that the [GUI] uses.

With this object, the GUI can [reflect the current property values] and [validate (changes/amendments)] to the
[template], so that it can [export the properties to a file].
```

```
    Class VmTemplateItem
    {
        [UInt32]     $Index
        [Guid]        $Guid
        [String]      $Name
        [Object]      $Role
        [String]      $Base
        [Object]    $Memory
        [Object]       $Hdd
        [UInt32]       $Gen
        [UInt32]      $Core
        [String]  $SwitchId
        [Object]     $Image
        VmTemplateItem(
        [UInt32]     $Index,
        [String]      $Name,
        [Object]      $Role,
        [String]      $Path,
        [Object]       $Ram,
        [Object]       $Hdd,
        [UInt32]       $Gen,
        [UInt32]      $Core,
        [String]    $Switch,
        [Object]     $Image)
        {
            $This.Index    = $Index
            $This.Guid     = $This.NewGuid()
            $This.Name     = $Name
            $This.Role     = $Role
            $This.Base     = $Path
            $This.Memory   = $Ram
            $This.Hdd      = $Hdd
            $This.Gen      = $Gen
            $This.Core     = $Core
            $This.SwitchId = $Switch
            $This.Image    = $Image
        }
        [Object] NewGuid()
        {
            Return [Guid]::NewGuid()
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Template]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Template.Output[0].GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    VmTemplateItem                           System.Object

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

Index    : 0
Guid     : 0cd57e77-4251-41db-8254-7f2da1a07050
Name     : desktop01
Role     : Client
Base     : C:\VDI
Memory   : 4.00 GB
Hdd      : 64.00 GB
Gen      : 2
Core     : 2
SwitchId : External
Image    : C:\Images\Win11_22H2_English_x64v1.iso

PS Prompt:\>
```

```
                                                                                                       ________________________/
\_____________________________________________________________________________________________________/ Class [VmTemplateItem]
  Class [VmTemplateFile] /----------------------------------------------------------------------------------------------------\
/------------------------
```

This is it. This is the point of the GUI and the function, exporting a desired virtual machine with cookie cutter properties and values so that it can be reinstantiated whenever needed, and controlled via the GUI or the CLI.

This is meant to [accelerate] the process of creating [new types] to [test software] or [operating system configurations]. In a weird way, it even provides a means of securing lab environments by allowing the machines to be [recreated] and [destroyed] as needed.

```
    Class VmTemplateFile
    {
        [String]      $Name
        [String]      $Role
        [Guid]        $Guid
        [Object]   $Account
        [Object]     $Image
        [String] $IpAddress
        [String]    $Domain
        [String]   $NetBios
        [String]   $Trusted
        [UInt32]    $Prefix
        [String]   $Netmask
        [String]   $Gateway
        [String[]]     $Dns
        [Object]      $Dhcp
        [String]      $Base
        [UInt64]    $Memory
        [UInt64]       $Hdd
        [UInt32]       $Gen
        [UInt32]      $Core
        [String]  $SwitchId
        VmTemplateFile([Object]$Template,[Object]$Account,[Object]$Network)
        {
            $This.Name      = $Template.Name
            $This.Role      = $Template.Role
            $This.Guid      = $Template.Guid
            $This.Account   = $Account
            $This.Image     = $Template.Image
            $This.IpAddress = $Network.IPAddress
            $This.Domain    = $Network.Domain
            $This.NetBios   = $Network.NetBios
            $This.Trusted   = $Network.Trusted
            $This.Prefix    = $Network.Prefix
            $This.Netmask   = $Network.Netmask
            $This.Gateway   = $Network.Gateway
            $This.Dns       = $Network.Dns
            $This.Dhcp      = $Network.Dhcp
            $This.Base      = $Template.Base
            $This.Memory    = $Template.Memory.Bytes
            $This.Hdd       = $Template.Hdd.Bytes
            $This.Gen       = $Template.Gen
            $This.Core      = $Template.Core
            $This.SwitchId  = $Template.SwitchId
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[File]>"
        }
    }
```

The given example to [instantiate] the object is a bit long-winded.
However, it is meant to pull all of the info it needs without replicating values that could cause a conflicts.

```
PS Prompt:\> $Ctrl.Template.VmTemplateFile($Ctrl.Template.Output[0],$Ctrl.Credential.Output,
$Ctrl.Template.VmTemplateNetwork($Ctrl.Master.Network))

Name      : desktop01
Role      : Client
Guid      : 0cd57e77-4251-41db-8254-7f2da1a07050
Account   : {<FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>}
Image     : C:\Images\Win11_22H2_English_x64v1.iso
IpAddress : 192.168.42.1
Domain    : securedigitsplus.com
NetBios   : SECURED
Trusted   : 192.168.42.2
Prefix    : 24
Netmask   : 255.255.255.0
Gateway   : 192.168.42.129
Dns       : {192.168.42.129}
Dhcp      : <FEVirtual.VmNetwork[Dhcp]>
Base      : C:\VDI
Memory    : 4294967296
Hdd       : 68719476736
Gen       : 2
Core      : 2
SwitchId  : External

PS Prompt:\>
```

```
                                                                                                           ________________________/
\_________________________________________________________________________________________________________/ Class [VmTemplateFile]
  Class [VmTemplateMaster] /---------------------------------------------------------------------------------------------------\
/-------------------------
```

This class is meant to [control] the above [template types]. With it, it is able to act as the [control mechanism] for the above [factory class types], as well as merging a bunch of [different objects] into the [output file].

```
    Class VmTemplateMaster
    {
        [Object] $Output
        VmTemplateMaster()
        {
            $This.Clear()
        }
        Clear()
        {
            $This.Output = @( )
        }
        [Object] VmTemplateFile([Object]$Template,[Object]$Accounts,[Object]$Node)
        {
            Return [VmTemplateFile]::New($Template,$Accounts,$Node)
        }
        [Object] VmTemplateNetwork([Object]$Network)
        {
            Return [VmTemplateNetwork]::New($Network)
        }
        [Object] VmTemplateItem(
        [UInt32]   $Index,
        [String]    $Name,
        [Object]    $Type,
        [String]    $Path,
        [Object]     $Ram,
        [Object]     $Hdd,
        [UInt32]     $Gen,
        [UInt32]    $Core,
        [String]  $Switch,
        [Object]   $Image)
        {
            Return [VmTemplateItem]::New($Index,
                                        $Name,
                                        $Type,
                                        $Path,
                                        $Ram,
                                        $Hdd,
                                        $Gen,
                                        $Core,
                                        $Switch,
                                        $Image)
        }
        [Object] VmRole([UInt32]$Index)
        {
            Return [VmRole]::New($Index)
        }
        [Object] VmByteSize([String]$Name,[UInt32]$Size)
        {
            Return [VmByteSize]::New($Name,$Size * 1GB)
        }
        Add(
        [String]    $Name,
        [UInt32]    $Type,
        [String]    $Path,
        [UInt32]     $Ram,
        [UInt32]     $Hdd,
        [UInt32]     $Gen,
        [UInt32]    $Core,
        [String]  $Switch,
        [Object]   $Image)
        {
            If ($Name -in $This.Output.Name)
            {
                Throw "Item already exists"
            }

            $This.Output += $This.VmTemplateItem($This.Output.Count,
            $Name,
            $This.VmRole($Type),
            $Path,
            $This.VmByteSize("Memory",$Ram),
            $This.VmByteSize("Drive",$Hdd),
            $Gen,
            $Core,
            $Switch,
            $Image)
        }
        Export([String]$Path,[Object]$Network,[Object]$Account,[UInt32]$Index)
        {
            If ($Index -gt $This.Output.Count)
            {
                Throw "Invalid index"
            }

            $Template      = $This.Output[$Index]
            $FilePath      = "{0}\{1}.fex" -f $Path, $Template.Name
            $Node          = $This.VmTemplateNetwork($Network)
            $Item          = $Network.Hosts | ? IPAddress -eq $Node.IPAddress
            $Item.Hostname = $Template.Name
            $Value         = $This.VmTemplateFile($Template,$Account,$Node)

            Export-CliXml -Path $FilePath -InputObject $Value -Depth 3

            If ([System.IO.File]::Exists($FilePath))
            {
                [Console]::WriteLine("Exported  [+] File: [$FilePath]")
            }
            Else
            {
                Throw "Something failed... bye."
            }
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmTemplate[Master]>"
        }
    }
```

```
                                                                                             __________________________/
\___________________________________________________________________________________________/ Class [VmTemplateMaster]
  Class [VmNodeDhcp] /-------------------------------------------------------------------------------------------------\
/--------------------
```

When the [template file] is returned into an object, this object is meant to reinstantiate the [Dhcp properties]. It may actually be unnecessary because it has no methods. However, that could easily be extended here, to add [Dhcp options] and other exclusions from existing (DHCP/DNS) servers or et cetera.

```
    Class VmNodeDhcp
    {
        [String]        $Name
        [String]  $SubnetMask
        [String]     $Network
        [String]  $StartRange
        [String]    $EndRange
        [String]   $Broadcast
        [String[]] $Exclusion
        VmNodeDhcp([Object]$Dhcp)
        {
            $This.Name       = $Dhcp.Name
            $This.SubnetMask = $Dhcp.SubnetMask
            $This.Network    = $Dhcp.Network
            $This.StartRange = $Dhcp.StartRange
            $This.EndRange   = $Dhcp.EndRange
            $This.Broadcast  = $Dhcp.Broadcast
            $This.Exclusion  = $Dhcp.Exclusion
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Dhcp]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Node.Template[0].Dhcp.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    VmNodeDhcp                               System.Object

PS Prompt:\> $Ctrl.Node.Template[0].Dhcp

Name       : 192.168.42.0/24
SubnetMask : 255.255.255.0
Network    : 192.168.42.0
StartRange : 192.168.42.1
EndRange   : 192.168.42.254
Broadcast  : 192.168.42.255
Exclusion  : {192.168.42.2, 192.168.42.129}

PS Prompt:\>
```

```
                                                                                                               ____________________/
\______________________________________________________________________________________________________________/ Class [VmNodeDhcp]
  Class [VmNodeSecurity] /--------------------------------------------------------------------------------------------------------\
/------------------------
```

```
This is strictly meant for containing [VmSecurity] settings such as [TPM], [shielded VM], [key protectors], etc.
```

```
    Class VmNodeSecurity
    {
        Hidden [String]  $Name
        [Object]     $Property
        [Object] $KeyProtector
        VmNodeSecurity([String]$Name)
        {
            $This.Name          = $Name
            $This.Refresh()
        }
        Refresh()
        {
            $This.Property     = Get-VmSecurity $This.Name -EA 0
            $This.KeyProtector = Get-VmKeyProtector -VmName $This.Name -EA 0
        }
        [Void] SetVmKeyProtector()
        {
            If ($This.KeyProtector.Length -le 4)
            {
                Set-VmKeyProtector -VmName $This.Name -NewLocalKeyProtector -Verbose
                $This.Refresh()
            }
        }
        ToggleTpm()
        {
            $This.Refresh()
            If ($This.KeyProtector.Length -le 4)
            {
                $This.SetVmKeyProtector()
            }

            Switch ([UInt32]$This.Property.TpmEnabled)
            {
                0
                {
                    Enable-VmTpm -VmName $This.Name -EA 0
                }
                1
                {
                    Disable-VmTpm -VmName $This.Name -EA 0
                }
            }

            $This.Refresh()
        }
```

```
  }
```

```
PS Prompt:\> $Vm.Security

Property   KeyProtector
--------   ------------
VMSecurity {0, 0, 20, 47...}

PS Prompt:\> $Vm.Security.Property

TpmEnabled                        : True
KsdEnabled                        : False
Shielded                          : False
EncryptStateAndVmMigrationTraffic : False
VirtualizationBasedSecurityOptOut : False
BindToHostTpm                     : False
CimSession                        : CimSession: .
ComputerName                      : L420-X64
IsDeleted                         : True

PS Prompt:\> [Char[]]$Vm.Security.KeyProtector -join ''
```

```
¶/ï»¿<?xml version="1.0" encoding="utf-8"?>
<Protector xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.microsoft.com/kps/2014/07">
  <Wrappings>
    <Wrapping>
      <Id>0</Id>
      <SigningCertificate>
MIIDGjCCAgKgAwIBAgIQHJjuPtthcKVJHN7QwpvKSDANBgkqhkiG9w0BAQsFADBJMUcwRQYDVQQDEz5TaGllbGRlZCBWTSBTaWduaW5nIENlcnR
pZmljYXRlIChVbnRydXN0ZWRHdWFyZGlhbikgKGw0MjAteDY0KTAeFw0yMzA0MTMxNDQ3MjJaFw0zMzA0MTMxNDQ3MjJaMEkxRzBFBgNVBAMTPl
NoaWVsZGVkIFZNIFNpZ25pbmcgQ2VydGlmaWNhdGUgKFVudHJ1c3RlZEd1YXJkaWFuKSAobDQyMC14NjQpMIIBIjANBgkqhkiG9w0BAQEFAAOCA
Q8AMIIBCgKCAQEA1KIZsRIvJ1kInDA/qTog3NM+zwXm5sYOUHICB3gM38nBeqdmEw1hK8mA1fdiuOtDkZRRLu2Tt1r/8FcZg4xM/k6eeHF0DkTi
MY3WFT1HC1ROMoIBpjPmoVVqijXzmI9HP2ex9oAe3mNTP8x+vb9n448KbLosqgDnEZwSTq/lSUF28GdjbbyzW6VzmXwv9/hb9FZFSNDPDdHk9M9
hk8m9TvoLiWIOOesuL40ScSggAALbJ8pj2awKBeNYnmRq0KthNm3W5nwJz/V4RpZUEcVyFkBHCye9YnB1Tphr2lf9nLbdZAEvDT9kx2NwHva4Mz
mqqgxuXVe233bOonTDNJ+VMQIDAQABMA0GCSqGSIb3DQEBCwUAA4IBAQCzmIKNS7kWcGXXX2DF+c6x1n08hwrFihZgdgBkLtkAIGuMVkHt/eZNZ
NFI7CELVoVdUkxtkvyLnx7d+v3ZeKMagm/B9s5yuc5zGlmbeSIe6hUb2WFFcvhntgzst/yzuXyPDrBY3gJoKU1R+DWRxtcc2isOZnrdBSV+JrQ8
hN9/v5JIiaRAMH7Gjsp2NXE4Eu9LWJpITWaBb8LkzST5ve1trRgADf/M8tut+vxIFRjasEDUcywZVhT7Q+RWqGo7VlKYqnspgG3/e/3o9x/m0eC
wweBb/VpomsvtATmB0sU34ojdT9xWykRPM6YADgHIhPDQa9v2NYhW/D+37lax15Rk
      </SigningCertificate>
      <SigningCertificateSignature ParentWrappingId="0">
        <Signature Algorithm="http://schemas.microsoft.com/kps/2014/07#rsa-pss-sha256">
          <SignatureValue>
O5DGTGxVSjwwtqPESbtRHovK9I+HtR4p3ZqT9/g5vMSlB1IWkLsE8eC1ix5gEmW3oueGS9HjXTbXbLOSbW74W+oY5A2LUa4WGEs37bAPz10C02k
qG+aQ8q7mHiL4dg8k0GGaN8BqKYiO2asivVLDh2LDBNasKyrJTbxOl4SVcf+9kgrWHiLrGUNS9Hkvg2VQHk8M6pOldw02WrPhCd3RdrfLIJicrh
2J1X20kHP0ROlwHoY42T3HZ/EktYqUhqWjrDg49wRqU1mxr24m2gQan3R/YisQy5V9n25OdsTShT2abjb7ue99f5MFAwcsWfr82vVUqdPUR5kJb
YqeRYz3IQ==
          </SignatureValue>
        </Signature>
      </SigningCertificateSignature>
      <EncryptionCertificate>
MIIDIDCCAgigAwIBAgIQGvxgVcY+2LRLG3oQ9sPDGjANBgkqhkiG9w0BAQsFADBMMUowSAYDVQQDE0FTaGllbGRlZCBWTSBFbmNyeXB0aW9uIEN
lcnRpZmljYXRlIChVbnRydXN0ZWRHdWFyZGlhbikgKGw0MjAteDY0KTAeFw0yMzA0MTMxNDQ3MjJaFw0zMzA0MTMxNDQ3MjJaMEwxSjBIBgNVBA
MTQVNoaWVsZGVkIFZNIEVuY3J5cHRpb24gQ2VydGlmaWNhdGUgKFVudHJ1c3RlZEd1YXJkaWFuKSAobDQyMC14NjQpMIIBIjANBgkqhkiG9w0BA
QEFAAOCAQ8AMIIBCgKCAQEAtFq+zjRGuja9Sbq1LGTqpFBYWg1/CDAG/LBx1tnC6L3eGUIKC0zNNpvyqtsOkwN3x5PdjTfy0lJ4K3jFmfF+ST9r
mW+1dRBNxwszfNFaH+A6Jsh0lAcpQYIH+GCov0URNDaENWd9p+na5b5fLfD8+FTI2oSWa12KarsAr7LKUDg3ruMW9gMJr8yo04EHMoJ7Tq/V/0f
EpmoyIuN6CQKVYXfEknHabkFSwKEyhHd+u3svgYRjhnVk3l8QKkbs6JLSeRb+CItIIdQok+Pfwu6wBJp07OyQeSE0fU+T6a60b9hddbBj461Byc
JKKOBaWAwboJmz1mUVuNl7xYdDMPNfuQIDAQABMA0GCSqGSIb3DQEBCwUAA4IBAQAZGsz9Wcb42QNYfRghg7jUR30juqKOkK8orZ/lSbtf3Q03/
bBSsglHCl6TZnFrQT/8eUIIXgIZ/leofbh+eUJG0bhxK9X8jxbdZOnp+qMVZ7u1q4ndI2DlWC+PfzgdOn69oBoyAGasAqy/FMRhtTqpPOBwmqYT
6oc0JJQZaJaj8wrUdPMSKEFCQh45DDSDeBiAXEWhjdsyqOmjuK+JVGo4SNgv1yGtHCmUQnbxp3OEqkaWNYomkNgKCy2GLJ474vazoXmWSoYab3y
CcEb5XdKuQc39L95AltjqyOuYm+az1/s1JtOw486HcOhnAVXDcL2tLWxvMmqdIxvK0uKZ1uTv
      </EncryptionCertificate>
      <EncryptionCertificateSignature>
        <Signature Algorithm="http://schemas.microsoft.com/kps/2014/07#rsa-pss-sha256">
          <SignatureValue>
AD8q/ppztZyoKUecwI8EIxMg+cwd6DVSj+KKrcBkkBpwpeGjFuDLr86QVAbrUVuYAR2MlRw17N3oof7fRQmdaj5aP+ZpphTVP4gqe0KLcpiHGUB
o3TAZ9aRqosuLrvrHkcvawaQYFqMfI128MD7r2MLuVLH9dA5obT9sK42DABKLYfU+6bZ/MDkBLVeyLVvwIwik835dANjFNNhfivQQqGfvQV1MFk
q8Jg+v/zenR6NmVivcRKrAPEJTW8zLrZSAxdqjl9KJOS4syZOJGiWqKT8xGYHMDg1gciOZN55kOmx/vwvkEb38LifXwJvv8pg90SPVk8EIcst6Q
6s7LUUNSA==
          </SignatureValue>
        </Signature>
      </EncryptionCertificateSignature>
      <TransportKey>
        <EncryptedData Algorithm="http://schemas.microsoft.com/kps/2014/07#rsa-oaep-mgf1-sha256">
          <CipherValue>
cWx79670ZiRVP0RMjVjXO7KBs9XKJ6D/vm8N65/D6LEXlIbZCOVYKzeTcXV3NZ9v4DlO46KAMFa3ZJDbb9WhuEmflt58Mx/sRRUrvIJq7uxGXEG
eqG8UQ7EiefmqO4jUB+yUH1AgDEAdLuWJk9TLwdyF+BmPskYR+BblgyJnZS5LRaqH/cKxnjGvLKDDzml2QdqZYVaRooyeTcKTZmjnnBFgFbjFM+
78jzifaZIVW8HaVRQeoUIRKSRBDtO2rGyPVTBRm1rDnrvc9z3IjQRAWfn5Y4FXnGkHl1KKErgnSbicfIUuribgBkVh6VyPtOhAMKbnS31/AP7QR
vE4GFieEQ==
          </CipherValue>
        </EncryptedData>
      </TransportKey>
    </Wrapping>
  </Wrappings>
  <TransportKeySignature>
    <KeyDerivationMethod Algorithm="http://schemas.microsoft.com/kps/2014/07#sp800-108-ctr-hmac-kdf" />
    <Signature Algorithm="http://schemas.microsoft.com/kps/2014/07#hmac-sha256">
      <SignatureValue>GGgSR7ECmblazmM+DeoLOXXBaEfgvf9KqO7Vh8i6w6A=</SignatureValue>
    </Signature>
  </TransportKeySignature>
  <GuardianSignature WrappingId="0">
    <Signature Algorithm="http://schemas.microsoft.com/kps/2014/07#rsa-pss-sha256">
      <SignatureValue>
ELLariHXa2WXkW9xqMwpvaHv2FQlusHZC7ayk2WAM1ptVdePXeBiE/U2qSuRUvbdH22G9Xb9VlMuh7QikVDq3HiJWUwMMhgDN02eOu4KhXogyvT
o8n7bQxI6n/gEI9XZvNeSYSVGcf/ZqAPHSVKLs+U7//EJXcyHmrZkcn44nHFcJyIWseTs6cfM+ahq8OgLlr1MM7rxtXpIE89Ao4rU7NeZAMeMWN
gwDTKWbFoTY4Svs01uYuiywDWrsXwFJQzZ1Ie3FQ9omIiInngs9ues+SdlpJinEMS99U0KSLC9fsJFWkthX2aBUFjBscphuvy9oQV6NtR7Xrm3E
h6A6w7OIg==
      </SignatureValue>
    </Signature>
  </GuardianSignature>
</Protector>
PS Prompt:\>
```

```
                                                                                               ________________________/
\_____________________________________________________________________________________________/ Class [VmNodeSecurity]
  Class [VmNodeImageFile] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Deserializes the information for the [ImageFile] in the [exported template], may not be necessary.

```
    Class VmNodeImageFile
    {
        [UInt32]    $Index
        [String]     $Type
        [String]  $Version
        [String]     $Name
        [String] $Fullname
        VmNodeImageFile([Object]$File)
        {
            $This.Index    = $File.Index
            $This.Type     = $File.Type
            $This.Version  = $File.Version
            $This.Name     = $File.Name
            $This.Fullname = $File.Fullname
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNodeImage[File]"
        }
    }
```

```
PS Prompt:\> $Ctrl.Node.Template.Image.File

Index    : 0
Type     : Windows
Version  : 10.0.22621.525
Name     : Win11_22H2_English_x64v1.iso
Fullname : C:\Images\Win11_22H2_English_x64v1.iso
```

```
PS Prompt:\>
```

```
                                                                                                    ________________________/
\__________________________________________________________________________________________________/ Class [VmNodeImageFile]
  Class [VmNodeImageEdition] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Deserializes the information for the [ImageEdition] in the [exported template], may not be necessary.

```
Class VmNodeImageEdition
{
    [UInt32]          $Index
    [String]           $Type
    [String]        $Version
    [String]           $Name
    [String]    $Description
    [String]           $Size
    [String]   $Architecture
    [String] $DestinationName
    [String]          $Label
    VmNodeImageEdition([Object]$Edition)
    {
        $This.Index           = $Edition.Index
        $This.Type            = $Edition.Type
        $This.Version         = $Edition.Version
        $This.Name            = $Edition.Name
        $This.Description     = $Edition.Description
        $This.Size            = $Edition.Size
        $This.Architecture    = $Edition.Architecture
        $This.DestinationName = $Edition.DestinationName
        $This.Label           = $Edition.Label
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNodeImage[Edition]"
    }
}
```

```
PS Prompt:\> $Ctrl.Node.Template.Image.Edition

Index           : 6
Type            : Client
Version         : 10.0.22621.525
Name            : Windows 11 Pro
Description     : Windows 11 Pro
Size            : 15.35 GB
Architecture    : 64
DestinationName : Windows 11 Pro (x64)
Label           : 11PRO64-10.0.22621.525

PS Prompt:\>
```

```
                                                                                                 ___________________________/
\_______________________________________________________________________________________________/ Class [VmNodeImageEdition]
  Class [VmNodeImageObject] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Deserializes the information for the [ImageObject] in the [exported template], may not be necessary.

```
Class VmNodeImageObject
{
    [Object] $File
    [Object] $Edition
    VmNodeImageObject([Object]$Image)
    {
        $This.File        = $This.VmNodeImageFile($Image.File)
```

```
            If ($Image.Edition)
            {
                $This.Edition = $This.VmNodeImageEdition($Image.Edition)
            }
        }
        [Object] VmNodeImageFile([Object]$File)
        {
            Return [VmNodeImageFile]::New($File)
        }
        [Object] VmNodeImageEdition([Object]$Edition)
        {
            Return [VmNodeImageEdition]::New($Edition)
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNodeImage[Object]"
        }
    }
```

```
PS Prompt:\> $Ctrl.Node.Template.Image

File                          Edition
----                          -------
<FEVirtual.VmNodeImage[File] <FEVirtual.VmNodeImage[Edition]

PS Prompt:\>
```

```
                                                                                             ___________________________/
\___________________________________________________________________________________________/ Class [VmNodeImageObject]
  Class [VmNodeTemplate] /---------------------------------------------------------------------------------------------\
/------------------------
```

This imports the [node template from file], and instantiates it into the [control object] for the [VmControllerMaster] object to create the corresponding [VmNodeObject].

```

    Class VmNodeTemplate
    {
        [UInt32]     $Index
        [Guid]        $Guid
        [String]      $Name
        [Object]      $Role
        [Object]   $Account
        [String] $IPAddress
        [String]    $Domain
        [String]   $NetBios
        [String]   $Trusted
        [UInt32]    $Prefix
        [String]   $Netmask
        [String]   $Gateway
        [String[]]     $Dns
        [Object]      $Dhcp
        [String]      $Base
        [Object]    $Memory
        [Object]       $Hdd
        [UInt32]       $Gen
        [Uint32]      $Core
        [String]  $SwitchId
        [Object]     $Image
        VmNodeTemplate([UInt32]$Index,[Object]$File)
        {
            $Item           = Import-CliXml -Path $File.Fullname
            $This.Index     = $Index
            $This.Name      = $Item.Name
            $This.Guid      = $Item.Guid
            $This.Role      = $Item.Role
            $This.Account   = $Item.Account
            $This.IPAddress = $Item.IPAddress
            $This.Domain    = $Item.Domain
            $This.NetBios   = $Item.NetBios
            $This.Trusted   = $Item.Trusted
            $This.Prefix    = $Item.Prefix
            $This.Netmask   = $Item.Netmask
            $This.Gateway   = $Item.Gateway
            $This.Dns       = $Item.Dns
            $This.Dhcp      = $This.VmNodeDhcp($Item.Dhcp)
            $This.Base      = $Item.Base
            $This.Memory    = $Item.Memory
            $This.Hdd       = $Item.Hdd
            $This.Gen       = $Item.Gen
            $This.Core      = $Item.Core
            $This.SwitchId  = $Item.SwitchId
            $This.Image     = $This.VmNodeImageObject($Item.Image)
        }
        [Object] NewGuid()
        {
            Return [Guid]::NewGuid()
        }
        [Object] VmNodeDhcp([Object]$Dhcp)
        {
            Return [VmNodeDhcp]::New($Dhcp)
        }
        [Object] VmNodeImageObject([Object]$Image)
        {
            Return [VmNodeImageObject]::New($Image)
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Template]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Node.Template

Index     : 0
Guid      : 705cdce0-3c62-492b-9b2f-659e5c7166c0
Name      : desktop01
Role      : Client
Account   : {<FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>}
IPAddress : 192.168.42.1
Domain    : securedigitsplus.com
NetBios   : SECURED
Trusted   : 192.168.42.2
Prefix    : 24
Netmask   : 255.255.255.0
Gateway   : 192.168.42.129
Dns       : {192.168.42.129}
Dhcp      : <FEVirtual.VmNode[Dhcp]>
Base      : C:\VDI
Memory    : 4294967296
Hdd       : 68719476736
Gen       : 2
Core      : 2
SwitchId  : External
Image     : <FEVirtual.VmNodeImage[Object]>

PS Prompt:\>
```

```
                                                                                                         ________________________/
\_______________________________________________________________________________________________________/ Class [VmNodeTemplate]
  Class [VmNodeItem] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

I believe that this item is deprecated, or unused.

```
    Class VmNodeItem
    {
```

```
    [UInt32]      $Index
    [Guid]         $Guid
    [Object]       $Name
    [Object]     $Memory
    [Object]       $Path
    [Object]        $Vhd
    [Object]    $VhdSize
    [Object] $Generation
    [UInt32]       $Core
    [Object] $SwitchName
    [Object]    $Network
    VmNodeItem([Object]$Node)
    {
        $This.Index      = $Node.Index
        $This.Guid       = $This.NewGuid()
        $This.Name       = $Node.Name
        $This.Memory     = $This.VmByteSize("Memory",$Node.Memory)
        $This.Path       = $Node.Base, $Node.Name -join '\'
        $This.Vhd        = "{0}\{1}\{1}.vhdx" -f $Node.Base, $Node.Name
        $This.VhdSize    = $This.VmByteSize("HDD",$Node.HDD)
    }
    [Object] NewGuid()
    {
        Return [Guid]::NewGuid()
    }
    [Object] VmByteSize([String]$Name,[UInt64]$Bytes)
    {
        Return [VmByteSize]::New($Name,$Bytes)
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNode[Item]>"
    }
}
```

```
                                                                                                                    ____________________/
\__________________________________________________________________________________________________________________/ Class [VmNodeItem]
  Class [VmNodeSwitch] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Strictly meant for the GUI to handle currently existing [virtual switches], so new switches may be made if needed from the GUI, or even the CLI.

```
Class VmNodeSwitch
{
    [UInt32]         $Index
    [Guid]            $Guid
    Hidden [Object] $Object
    [String]          $Name
    [String]          $Type
    [String]   $Description
    VmNodeSwitch([UInt32]$Index,[Object]$Object)
    {
        $This.Index       = $Index
        $This.Guid        = $Object.Id
        $This.Object      = $Object
        $This.Name        = $Object.Name
        $This.Type        = $Object.SwitchType
        $This.Description = $Object.NetAdapterInterfaceDescription
    }
    [Object] NewGuid()
    {
        Return [Guid]::NewGuid()
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNode[Switch]>"
    }
}
```

```
PS Prompt:\> $Ctrl.Node.Switch[0]

Index       : 0
Guid        : e298bb2e-6a4a-4af7-9e2a-c0ce0f36eede
Name        : External
Type        : External
Description : SAMSUNG Mobile USB Remote NDIS Network Device

PS Prompt:\>
```

```
                                                                                                 ______________________/
\_______________________________________________________________________________________________/ Class [VmNodeSwitch]
  Class [VmNodeHost] /-------------------------------------------------------------------------------------------------\
/--------------------
```

```
Meant to show any [existing virtual machines], so that [new templates] do not conflict with them.
```

```

    Class VmNodeHost
    {
        [UInt32]      $Index
        [Guid]         $Guid
        [Object]       $Name
        [Object]     $Memory
        [Object]       $Path
        [Object]        $Vhd
        [Object]    $VhdSize
        [Object] $Generation
        [UInt32]       $Core
        [Object] $SwitchName
        VmNodeHost([UInt32]$Index,[Object]$Node)
        {
            $This.Index      = $Node.Index
            $This.Guid       = $Node.Id
            $This.Name       = $Node.Name
            $This.Memory     = $This.Size("Memory",$Node.MemoryStartup)
            $This.Path       = $Node.Path
            $This.Vhd        = $Node.HardDrives[0].Path
            $This.VhdSize    = $This.Size("HDD",$This.Drive())
            $This.Generation = $Node.Generation
            $This.Core       = $Node.ProcessorCount
            $This.SwitchName = $Node.NetworkAdapters[0].SwitchName
        }
        [UInt64] Drive()
        {
            Return Get-Item $This.Vhd | % Length
        }
        [Object] Size([String]$Name,[UInt64]$SizeBytes)
        {
            Return [VmByteSize]::New($Name,$SizeBytes)
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Host]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Node.Host

Index      : 0
Guid       : d6c5d9df-b0b8-4498-b8c0-5815824e8c1a
Name       : desktop01
Memory     : 4.00 GB
```

```
Path       : C:\VDI\desktop01\desktop01
Vhd        : C:\VDI\desktop01\desktop01.vhdx
VhdSize    : 4.00 MB
Generation : 2
Core       : 2
SwitchName : External

PS Prompt:\>
```

```
                                                                                                    ____________________/
\__________________________________________________________________________________________________/ Class [VmNodeHost]
  Class [VmNodeSlot] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

This object is meant for the GUI, and it's purpose is to [differentiate between existing hosts] and [non-existent template files]. At some point, [further integration] between these [two objects] will be implemented, where a [host] and a [template] can be used to provide extended control capabilities, or be used like a “key”.

For the time being, that will be [strictly handled by the CLI].

```
    Class VmNodeSlot
    {
        [String] $Index
        [Guid]     $Guid
        [String]  $Name
        [String]  $Type
        VmNodeSlot([UInt32]$Index,[Object]$Node)
        {
            $This.Index      = $Index
            $This.Guid       = $Node.Guid
            $This.Name       = $Node.Name
            $This.Type       = Switch -Regex ($Node.GetType().Name)
            {
                "VmNodeHost"     { "Host"     }
                "VmNodeTemplate" { "Template" }
            }
        }
    }
```

```
PS Prompt:\> $Ctrl.Node.Object

Index Guid                                 Name      Type
----- ----                                 ----      ----
0     d6c5d9df-b0b8-4498-b8c0-5815824e8c1a desktop01 Host
1     705cdce0-3c62-492b-9b2f-659e5c7166c0 desktop01 Template

PS Prompt:\>
```

```
                                                                                                    ____________________/
\__________________________________________________________________________________________________/ Class [VmNodeSlot]
  Class [VmNodeScriptBlockLine] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Meant to provide granular control over the script block entries.

```
    Class VmNodeScriptBlockLine
    {
        [UInt32] $Index
        [String]  $Line
        VmNodeScriptBlockLine([UInt32]$Index,[String]$Line)
        {
            $This.Index = $Index
            $This.Line  = $Line
        }
        [String] ToString()
        {
            Return $This.Line
        }
    }
```

```
PS Prompt:\> $Vm.Script.Output[0].Content[0]

Index Line
----- ----
    0 # Set persistent information

PS Prompt:\>
```

```
                                                                                                    ______________________________/
\___________________________________________________________________________________________________/ Class [VmNodeScriptBlockLine]
  Class [VmNodeScriptBlockItem] /------------------------------------------------------------------------------------------------\
/------------------------------
```

Provides extended control over the flow of (*configuration/execution*) scripts.

At some point, there will be additional controls that dictate whether a script can be handled via [running], or [transmitting].

[Running] the script will use the [MSVM_Keyboard] object to pass through [every individual character].
[Transmitting] the script will use a [TCP Session] object to expedite the process of [configuration].

```
    Class VmNodeScriptBlockItem
    {
        [UInt32]       $Index
        [UInt32]       $Phase
        [String]        $Name
        [String] $DisplayName
        [Object]     $Content
        [UInt32]    $Complete
        VmNodeScriptBlockItem([UInt32]$Index,[UInt32]$Phase,[String]$Name,[String]$DisplayName,
        [String[]]$Content)
        {
            $This.Index       = $Index
            $This.Phase       = $Phase
            $This.Name        = $Name
            $This.DisplayName = $DisplayName

            $This.Load($Content)
        }
        Clear()
        {
            $This.Content     = @( )
        }
        Load([String[]]$Content)
        {
            $This.Clear()
            $This.Add("# $($This.DisplayName)")

            ForEach ($Line in $Content)
            {
                $This.Add($Line)
            }

            $This.Add('')
        }
        [Object] VmNodeScriptBlockLine([UInt32]$Index,[String]$Line)
        {
            Return [VmNodeScriptBlockLine]::New($Index,$Line)
        }
        Add([String]$Line)
        {
            $This.Content += $This.VmNodeScriptBlockLine($This.Content.Count,$Line)
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNodeScriptBlock[Item]>"
        }
    }
```

```
PS Prompt:\> $Vm.Script.Output[0]

Index       : 0
Phase       : 1
Name        : SetPersistentInfo
DisplayName : Set persistent information
Content     : {# Set persistent information, $Root      = "HKLM:\Software\Policies\Secure Digits Plus LLC",
              $Name      = "desktop01", $Path      = "$Root\ComputerInfo"...}
Complete    : 0

PS Prompt:\>
```

```
                                                                                                    ______________________________/
\___________________________________________________________________________________________________/ Class [VmNodeScriptBlockItem]
  Class [VmNodeScriptBlockController] /--------------------------------------------------------------------------------\
/-------------------------------------
```

This orchestrates the process of [scripting] the [virtual machine] from the [template].
Though, to be clear, there's still plenty of work to do with it.

```
    Class VmNodeScriptBlockController
    {
        [UInt32] $Selected
        [UInt32]    $Count
        [Object]   $Output
        VmNodeScriptBlockController()
        {
            $This.Clear()
        }
        Clear()
        {
            $This.Output = @( )
            $This.Count  = 0
        }
        Reset()
        {
            ForEach ($Item in $This.Output)
            {
                $Item.Complete = 0
            }

            $This.Selected = 0
        }
        [Object] VmNodeScriptBlockItem([UInt32]$Index,[UInt32]$Phase,[String]$Name,[String]$DisplayName,
        [String[]]$Content)
        {
            Return [VmNodeScriptBlockItem]::New($Index,$Phase,$Name,$DisplayName,$Content)
        }
        Add([String]$Phase,[String]$Name,[String]$DisplayName,[String[]]$Content)
        {
            $This.Output += $This.VmNodeScriptBlockItem($This.Output.Count,$Phase,$Name,$DisplayName,
                                                        $Content)
            $This.Count   = $This.Output.Count
        }
        Select([UInt32]$Index)
        {
            If ($Index -gt $This.Count)
            {
                Throw "Invalid index"
            }

            $This.Selected = $Index
        }
        [Object] Current()
        {
            Return $This.Output[$This.Selected]
        }
        [Object] Get([String]$Name)
        {
            Return $This.Output | ? Name -eq $Name
        }
        [Object] Get([UInt32]$Index)
        {
            Return $This.Output | ? Index -eq $Index
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNodeScriptBlock[Controller]>"
        }
    }
```

```
PS Prompt:\> $Vm.Script

Selected Count Output
-------- ----- ------
       0    16 {<FEVirtual.VmNodeScriptBlock[Item]>, <FEVirtual.VmNodeScriptBlock[Item]>, <FEVirtual.VmNodeS...

PS Prompt:\>
```

```
                                                                                               ____________________________________/
\_____________________________________________________________________________________________/ Class [VmNodeScriptBlockController]
  Class [VmNodePropertyItem] /--------------------------------------------------------------------------------------------------\
/---------------------------
```

Simply meant to exfiltrate all of the properties that belong to the [virtual machine] object.

```
    Class VmNodePropertyItem
    {
        [UInt32] $Index
        [String]  $Name
        [Object] $Value
        VmNodePropertyItem([UInt32]$Index,[Object]$Property)
        {
            $This.Index = $Index
            $This.Name  = $Property.Name
            $This.Value = $Property.Value
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmProperty[Item]>"
        }
    }
```

```
PS Prompt:\> $Vm.Property.Output[0] | Format-List

Index : 0
Name  : ParentCheckpointId
Value :

PS Prompt:\>
```

```
                                                                                                         ____________________________/
\_______________________________________________________________________________________________________/ Class [VmNodePropertyItem]
  Class [VmNodePropertyList] /--------------------------------------------------------------------------------------------------\
/---------------------------
```

This contains the [entire list] of [properties] that belong to the [virtual machine].

```
Class VmNodePropertyList
{
    [String]   $Name
    [UInt32]  $Count
    [Object] $Output
    VmNodePropertyList()
    {
        $This.Name = "VmProperty[List]"
    }
    Clear()
    {
        $This.Output = @( )
    }
    [Object] VmNodePropertyItem([UInt32]$Index,[Object]$Property)
    {
        Return [VmNodePropertyItem]::($Index,$Property)
    }
    Add([Object]$Property)
    {
        $This.Output += $This.VmNodePropertyItem($This.Output.Count,$Property)
        $This.Count   = $This.Output.Count
    }
    [String] ToString()
    {
        Return "({0}) <FEVirtual.VmProperty[List]>" -f $This.Count
    }
}
```

```
PS Prompt:\> $Vm.Property.Output

Index Name                                 Value
----- ----                                 -----
    0 ParentCheckpointId
    1 ParentCheckpointName
    2 VMName                               desktop01
    3 VMId                                 d6c5d9df-b0b8-4498-b8c0-5815824e8c1a
    4 CheckpointFileLocation               C:\VDI\desktop01\desktop01
    5 ConfigurationLocation                C:\VDI\desktop01\desktop01
    6 SmartPagingFileInUse                 False
    7 SmartPagingFilePath                  C:\VDI\desktop01\desktop01
    8 SnapshotFileLocation                 C:\VDI\desktop01\desktop01
    9 AutomaticStartAction                 StartIfRunning
   10 AutomaticStartDelay                  0
   11 AutomaticStopAction                  Save
   12 AutomaticCriticalErrorAction         Pause
   13 AutomaticCriticalErrorActionTimeout  30
   14 AutomaticCheckpointsEnabled          True
   15 CPUUsage                             0
   16 MemoryAssigned                       0
   17 MemoryDemand                         0
   18 MemoryStatus
   19 NumaAligned
   20 NumaNodesCount                       1
   21 NumaSocketCount                      1
   22 Heartbeat
   23 IntegrationServicesState
   24 IntegrationServicesVersion           0.0
   25 Uptime                               00:00:00
   26 OperationalStatus                    {Ok}
   27 PrimaryOperationalStatus             Ok
   28 SecondaryOperationalStatus
   29 StatusDescriptions                   {Operating normally}
   30 PrimaryStatusDescription             Operating normally
   31 SecondaryStatusDescription
   32 Status                               Operating normally
   33 ReplicationHealth                    NotApplicable
   34 ReplicationMode                      None
   35 ReplicationState                     Disabled
   36 ResourceMeteringEnabled              True
   37 CheckpointType                       Standard
```

```
 38 EnhancedSessionTransportType        VMBus
 39 Groups                              {}
 40 Version                             9.0
 41 VirtualMachineType                  RealizedVirtualMachine
 42 VirtualMachineSubType               Generation2
 43 Notes
 44 State                               Off
 45 ComPort1                            VMComPort (Name = 'COM 1', VMName = 'desktop01') [Id = 'Microsoft:...
 46 ComPort2                            VMComPort (Name = 'COM 2', VMName = 'desktop01') [Id = 'Microsoft:...
 47 DVDDrives                           {}
 48 FibreChannelHostBusAdapters         {}
 49 FloppyDrive
 50 HardDrives                          {Hard Drive on SCSI controller number 0 at location 0}
 51 RemoteFxAdapter
 52 VMIntegrationService                {Guest Service Interface, Heartbeat, Key-Value Pair Exchange, Shut...
 53 DynamicMemoryEnabled                False
 54 MemoryMaximum                       1099511627776
 55 MemoryMinimum                       536870912
 56 MemoryStartup                       4294967296
 57 ProcessorCount                      2
 58 BatteryPassthroughEnabled           True
 59 Generation                          2
 60 IsClustered                         False
 61 ParentSnapshotId
 62 ParentSnapshotName
 63 Path                                C:\VDI\desktop01\desktop01
 64 SizeOfSystemFiles                   73728
 65 GuestControlledCacheTypes           False
 66 LowMemoryMappedIoSpace              134217728
 67 HighMemoryMappedIoSpace             536870912
 68 LockOnDisconnect                    Off
 69 CreationTime                        5/3/2023 5:34:58 PM
 70 Id                                  d6c5d9df-b0b8-4498-b8c0-5815824e8c1a
 71 Name                                desktop01
 72 NetworkAdapters                     {Network Adapter}
 73 CimSession                          CimSession: .
 74 ComputerName                        L420-X64
 75 IsDeleted                           False

PS Prompt:\>
```

```
                                                                                                    ____________________________/
\__________________________________________________________________________________________________/ Class [VmNodePropertyList]
 Class [VmNodeCheckpoint] /------------------------------------------------------------------------------------------------------\
/-------------------------
```

Meant to capture existing checkpoints for the [virtual machine].

```
Class VmNodeCheckpoint
{
    Hidden [Object] $Checkpoint
    [UInt32]              $Index
    [String]               $Name
    [String]               $Type
    [DateTime]             $Time
    VmNodeCheckPoint([UInt32]$Index,[Object]$Checkpoint)
    {
        $This.Checkpoint = $Checkpoint
        $This.Index      = $Index
        $This.Name       = $Checkpoint.Name
        $This.Type       = $Checkpoint.SnapshotType
        $This.Time       = $Checkpoint.CreationTime
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmCheckpoint>"
    }
}
```

```
PS Prompt:\> $Vm.Checkpoint

Index Name                                                      Type     Time
----- ----                                                      ----     ----
    0 Automatic Checkpoint - desktop01 - (5/3/2023 - 5:54:18 PM) Standard 5/3/2023 5:54:18 PM
    1 2023_0503-CheckPoint                                      Standard 5/3/2023 5:54:50 PM

PS Prompt:\>
```

```
                                                                                                        ___________________________/
\_________________________________________________________________________________________________________/ Class [VmNodeCheckpoint]
  Class [VmNodeNetwork] /----------------------------------------------------------------------------------------------------\
/-----------------------
```

Contains very similar information to the [VmTemplateNetwork] object, however it also has the [transmit port].
These settings are implemented into an [item property] on the [target system].

```
    Class VmNodeNetwork
    {
        [String]    $Domain
        [String]   $NetBios
        [String] $IPAddress
        [String]   $Network
        [String] $Broadcast
        [String]   $Trusted
        [UInt32]    $Prefix
        [String]   $Netmask
        [String]   $Gateway
        [String[]]     $Dns
        [Object]      $Dhcp
        [UInt32]  $Transmit
        VmNodeNetwork([Object]$Node)
        {
            $This.Domain    = $Node.Domain
            $This.NetBios   = $Node.NetBios
            $This.IPAddress = $Node.IpAddress
            $This.Network   = $Node.Dhcp.Network
            $This.Broadcast = $Node.Dhcp.Broadcast
            $This.Trusted   = $Node.Trusted
            $This.Prefix    = $Node.Prefix
            $This.Netmask   = $Node.Netmask
            $This.Gateway   = $Node.Gateway
            $This.Dns       = $Node.Dns
            $This.Dhcp      = $Node.Dhcp
            $This.Transmit  = @(13000,$Node.Transmit)[!!$Node.Transmit]
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Network]>"
        }
    }
```

```
PS Prompt:\> $Vm.Network

Domain    : securedigitsplus.com
NetBios   : SECURED
IPAddress : 192.168.42.1
Network   : 192.168.42.0
Broadcast : 192.168.42.255
Trusted   : 192.168.42.2
Prefix    : 24
Netmask   : 255.255.255.0
Gateway   : 192.168.42.129
Dns       : {192.168.42.129}
Dhcp      : <FEVirtual.VmNode[Dhcp]>
Transmit  : 13000
```

```
PS Prompt:\>
```

```
                                                                                                                       ________________________/
\______________________________________________________________________________________________/ Class [VmNodeNetwork]
  Class [VmNodeObject] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
Class VmNodeObject
{
    Hidden [Object]   $Object
    Hidden [UInt32]     $Mode
    [Object]         $Console
    [Object]            $Name
    [Object]            $Role
    [Object]          $Memory
    [Object]            $Path
    [Object]             $Vhd
    [Object]         $VhdSize
    [Object]      $Generation
    [UInt32]            $Core
    [Object]          $Switch
    [Object]        $Firmware
    [UInt32]          $Exists
    [Object]            $Guid
    [Object]         $Account
    [Object]         $Network
    [Object]           $Image
    [Object]          $Script
    [Object]      $Checkpoint
    Hidden [Object] $Security
    Hidden [Object] $Property
    Hidden [Object]  $Control
    Hidden [Object] $Keyboard
    VmNodeObject([Object]$Node)
    {
        # Meant to build a new VM
        $This.Mode       = 1
        $This.Role       = $Node.Role
        $This.StartConsole()

        $This.Name       = $Node.Name
        [Void]$This.Get()

        Switch ($This.Exists)
        {
            0
            {
                $This.Memory     = $This.Size("Ram",$Node.Memory)
                $This.Path       = "{0}\{1}" -f $Node.Base, $Node.Name
                $This.Vhd        = "{0}\{1}\{1}.vhdx" -f $Node.Base, $Node.Name
                $This.VhdSize    = $This.Size("Hdd",$Node.HDD)
                $This.Generation = $Node.Gen
                $This.Core       = $Node.Core
                $This.Switch     = @($Node.SwitchId)
            }
            1
            {
                $This.Memory     = $This.Size("Ram",$This.Object.MemoryStartup)
                $This.Path       = $This.Object.Path
                $xVhd            = Get-Vhd $This.Object.HardDrives[0].Path
                $This.Vhd        = @($xVhd.Path,$xVhd.ParentPath)[!!$xVhd.ParentPath]
                $This.VhdSize    = $xVhd.Size
                $This.Generation = $This.Object.Generation
                $This.Core       = $This.Object.ProcessorCount
                $This.Switch     = @($This.Object.NetworkAdapters[0].SwitchName)
            }
        }

        $This.Account    = $Node.Account
        $This.Network    = $This.VmNodeNetwork($Node)
        $This.Image      = $Node.Image
        $This.Script     = $This.VmNodeScriptBlockController()
        $This.Security   = $This.VmNodeSecurity()
    }
    StartConsole()
    {
        # Instantiates and initializes the console
        $This.Console = New-FEConsole
        $This.Console.Initialize()
        $This.Status()
    }
    Status()
    {
        # If enabled, shows the last item added to the console
        If ($This.Mode -gt 0)
        {
            [Console]::WriteLine($This.Console.Last())
        }
    }
    Update([Int32]$State,[String]$Status)
    {
        # Updates the console
        $This.Console.Update($State,$Status)
        $This.Status()
    }
    Error([String]$Status)
    {
        $This.Console.Update(-1,$Status)
    }
    DumpConsole()
    {
        $xPath = "{0}\{1}-{2}.log" -f $This.LogPath(), $This.Now(), $This.Name
        $This.Update(100,"[+] Dumping console: [$xPath]")
        $This.Console.Finalize()

        $Value = $This.Console.Output | % ToString

        [System.IO.File]::WriteAllLines($xPath,$Value)
    }
    [String] LogPath()
    {
        $xPath = $This.ProgramData()

        ForEach ($Folder in $This.Author(), "Logs")
        {
            $xPath = $xPath, $Folder -join "\"
            If (![System.IO.Directory]::Exists($xPath))
            {
                [System.IO.Directory]::CreateDirectory($xPath)
            }
        }

        Return $xPath
    }
    [String] Now()
    {
        Return [DateTime]::Now.ToString("yyyy-MMdd_HHmmss")
    }
    [Object] Wmi([String]$Type)
    {
        Return Get-WmiObject $Type -Namespace Root\Virtualization\V2
    }
    [Object] VmNodeNetwork([Object]$Node)
    {
        Return [VmNodeNetwork]::New($Node)
    }
    [Object] VmNodeCheckPoint([UInt32]$Index,[Object]$Checkpoint)
    {
        Return [VmNodeCheckPoint]::New($Index,$Checkpoint)
    }
    [Object] VmNodePropertyList()
```

```
        {
            Return [VmNodePropertyList]::New()
        }
        [Object] VmNodeScriptBlockController()
        {
            Return [VmNodeScriptBlockController]::New()
        }
        [Object] VmNodeSecurity()
        {
            Return [VmNodeSecurity]::New($This.Name)
        }
        [Object] Get()
        {
            $This.Object   = Get-VM -Name $This.Name -EA 0
            $This.Exists   = $This.Object.Count -gt 0
            $This.Guid     = @($Null,$This.Object.Id)[$This.Exists]

            Return @($Null,$This.Object)[$This.Exists]
        }
        [Object] Size([String]$Name,[UInt64]$SizeBytes)
        {
            Return [VmByteSize]::New($Name,$SizeBytes)
        }
        [String] Hostname()
        {
            Return [Environment]::MachineName
        }
        [String] ProgramData()
        {
            Return [Environment]::GetEnvironmentVariable("ProgramData")
        }
        [String] Author()
        {
            Return "Secure Digits Plus LLC"
        }
        [String] GuestName()
        {
            Return $This.Network.Hostname()
        }
        Connect()
        {
            $This.Update(0,"[~] Connecting : $($This.Name)")
            $Splat           = @{

                Filepath     = "vmconnect"
                ArgumentList = @($This.Hostname(),$This.Name)
                Verbose      = $True
                PassThru     = $True
            }

            Start-Process @Splat
        }
        New()
        {
            $Null = $This.Get()
            If ($This.Exists -ne 0)
            {
                $This.Error("[!] Exists : $($This.Name)")
            }

            $Splat                 = @{

                Name               = $This.Name
                MemoryStartupBytes = $This.Memory.Bytes
                Path               = $This.Path
                NewVhdPath         = $This.Vhd
                NewVhdSizeBytes    = $This.VhdSize.Bytes
                Generation         = $This.Generation
                SwitchName         = $This.Switch[0]
            }

            $This.Update(0,"[~] Creating : $($This.Name)")
```

```powershell
        # Verbosity level
        Switch ($This.Mode)
        {
            Default { New-VM @Splat }
            2       { New-VM @Splat -Verbose }
        }

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Set-VMMemory -VmName $This.Name -DynamicMemoryEnabled 0 }
            2       { Set-VMMemory -VmName $This.Name -DynamicMemoryEnabled 0 -Verbose }
        }

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Enable-VmResourceMetering -VmName $This.Name }
            2       { Enable-VmResourceMetering -VmName $This.Name -Verbose }
        }

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Set-Vm -Name $This.Name -CheckpointType Standard }
            2       { Set-Vm -Name $This.Name -CheckpointType Standard -Verbose -EA 0 }
        }

        $Item                  = $This.Get()
        $This.Firmware         = $This.GetVmFirmware()
        $This.SetVMProcessor()
        $This.Security.Refresh()

        $This.Script           = $This.VmNodeScriptBlockController()
        $This.Property         = $This.VmNodePropertyList()

        ForEach ($Property in $Item.PSObject.Properties)
        {
            $This.Property.Add($Property)
        }
    }
    Start()
    {
        $Vm = $This.Get()
        If (!$Vm)
        {
            $This.Error("[!] Exception : $($This.Name) [does not exist]")
        }

        ElseIf ($Vm.State -eq "Running")
        {
            $This.Error("[!] Exception : $($This.Name) [already started]")
        }

        Else
        {
            $This.Update(1,"[~] Starting : $($This.Name)")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { $Vm | Start-VM }
                2       { $Vm | Start-VM -Verbose }
            }
        }
    }
    Stop()
    {
        [Void]$This.Get()
        If (!$This.Object)
        {
            $This.Error("[!] Exception : $($This.Name) [does not exist]")
        }

        ElseIf ($This.Object.State -ne "Running")
        {
            $This.Error("[!] Exception : $($This.Name) [not running]")
        }

        Else
        {
            $This.Update(0,"[~] Stopping : $($This.Name)")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { $This.Get() | ? State -ne Off | Stop-VM -Force }
                2       { $This.Get() | ? State -ne Off | Stop-VM -Force -Verbose }
            }
        }
    }
    Reset()
    {
        $Vm = $This.Get()
        If (!$Vm)
        {
            $This.Error("[!] Exception : $($This.Name) [does not exist]")
        }

        ElseIf ($Vm.State -ne "Running")
        {
            $This.Error("[!] Exception : $($This.Name) [not running]")
        }

        Else
        {
            $This.Update(0,"[~] Restarting : $($This.Name)")
            $This.Stop()
            $This.Start()
            $This.Idle(5,5)
        }
    }
    Remove()
    {
        $Vm = $This.Get()
        If (!$Vm)
        {
            $This.Error("[!] Exception : $($This.Name) [does not exist]")
        }

        $This.Update(0,"[~] Removing : $($This.Name)")

        If ($Vm.State -ne "Off")
        {
            $This.Update(0,"[~] State : $($This.Name) [attempting shutdown]")
            Switch -Regex ($Vm.State)
            {
                "(^Paused$|^Saved$)"
                {
                    $This.Start()
                    Do
                    {
                        Start-Sleep 1
                    }
                    Until ($This.Get().State -eq "Running")
                }
            }

            $This.Stop()
            Do
            {
                Start-Sleep 1
            }
            Until ($This.Get().State -eq "Off")
        }

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { $This.Get() | Remove-VM -Confirm:$False -Force -EA 0 }
            2       { $This.Get() | Remove-VM -Confirm:$False -Force -Verbose -EA 0 }
        }

        $This.Firmware         = $Null
        $This.Exists           = 0

        $This.Update(0,"[~] Vhd  : [$($This.Vhd)]")

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Remove-Item $This.Vhd -Confirm:$False -Force -EA 0 }
            2       { Remove-Item $This.Vhd -Confirm:$False -Force -Verbose -EA 0 }
        }

        $This.Update(0,"[~] Path : [$($This.Path)]")
        ForEach ($Item in Get-ChildItem $This.Path -Recurse | Sort-Object -Descending)
        {
            $This.Update(0,"[~] $($Item.Fullname)")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Remove-Item $Item.Fullname -Confirm:$False -EA 0 }
                2       { Remove-Item $Item.Fullname -Confirm:$False -Verbose -EA 0 }
            }
        }

        $Parent = Split-Path $This.Path -Parent
        $Leaf   = Split-Path $Parent -Leaf
        If ($Leaf -eq $This.Name)
        {
            $This.Update(0,"[~] $($Item.Fullname)")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Remove-Item $Parent -Recurse -Confirm:$False -EA 0 }
                2       { Remove-Item $Parent -Recurse -Confirm:$False -Verbose -EA 0 }
            }
        }

        $This.Update(1,"[ ] Removed : $($Item.Fullname)")

        $This.DumpConsole()
    }
    GetCheckpoint()
    {
        $This.Update(0,"[~] Getting Checkpoint(s)")

        $This.Checkpoint = @( )
        $List            = Switch ($This.Mode)
        {
            Default { Get-VmCheckpoint -VMName $This.Name -EA 0 }
            2       { Get-VmCheckpoint -VMName $This.Name -Verbose -EA 0 }
        }

        If ($List.Count -gt 0)
        {
            ForEach ($Item in $List)
            {
                $This.Checkpoint += $This.VmCheckpoint($This.Checkpoint.Count,$Item)
            }
        }
    }
```

```
        NewCheckpoint()
        {
            $ID = "{0}-{1}" -f $This.Name, $This.Now()
            $This.Update(0,"[~] New Checkpoint [$ID]")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { $This.Get() | Checkpoint-Vm -SnapshotName $ID }
                2       { $This.Get() | Checkpoint-Vm -SnapshotName $ID -Verbose -EA 0 }
            }

            $This.GetCheckpoint()
        }
        RestoreCheckpoint([UInt32]$Index)
        {
            If ($Index -gt $This.Checkpoint.Count)
            {
                Throw "Invalid index"
            }

            $Item = $This.Checkpoint[$Index]

            $This.Update(0,"[~] Restoring Checkpoint [$($Item.Name)]")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Restore-VMCheckpoint -Name $Item.Name -VMName $This.Name -Confirm:0 -EA 0 }
                2       { Restore-VMCheckpoint -Name $Item.Name -VMName $This.Name -Confirm:0 -Verbose -EA 0}
            }
        }
        RestoreCheckpoint([String]$String)
        {
            $Item = $This.Checkpoint | ? Name -match $String

            If (!$Item)
            {
                Throw "Invalid entry"
            }
            ElseIf ($Item.Count -gt 1)
            {
                $This.Update(0,"[!] Multiple entries detected, select index or limit search string")

                $D = (([String[]]$Item.Index) | Sort-Object Length)[-1].Length
                $Item | % {

                    $Line = "({0:d$D}) [{1}]: {2}" -f $_.Index,
                                                      $_.Time.ToString("MM-dd-yyyy HH:mm:ss"), $_.Name
                    [Console]::WriteLine($Line)
                }
            }
            Else
            {
                $This.RestoreCheckpoint($Item.Index)
            }
        }
        RemoveCheckpoint([UInt32]$Index)
        {
            If ($Index -gt $This.Checkpoint.Count)
            {
                Throw "Invalid index"
            }

            $Item = $This.Checkpoint[$Index]

            $This.Update(0,"[~] Removing Checkpoint [$($Item.Name)]")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Remove-VMCheckpoint -Name $Item.Name -VMName $This.Name -Confirm:0 -EA 0 }
                2       { Remove-VMCheckpoint -Name $Item.Name -VMName $This.Name -Confirm:0 -Verbose -EA 0 }
            }

            $This.GetCheckpoint()
        }
        [Object] Measure()
        {
            If (!$This.Exists)
            {
                Throw "Cannot measure a virtual machine when it does not exist"
            }

            Return Measure-Vm -Name $This.Name
        }
        [String] GetRegistryPath()
        {
            Return "HKLM:\Software\Policies\Secure Digits Plus LLC"
        }
        [Object] GetVmFirmware()
        {
            $This.Update(0,"[~] Getting VmFirmware : $($This.Name)")
            $Item = Switch ($This.Generation)
            {
                1
                {
                    # Verbosity level
                    Switch ($This.Mode)
                    {
                        Default { Get-VmBios -VmName $This.Name }
                        2       { Get-VmBios -VmName $This.Name -Verbose }
                    }
                }
                2
                {
                    # Verbosity level
                    Switch ($This.Mode)
                    {
                        Default { Get-VmFirmware -VmName $This.Name }
                        2       { Get-VmFirmware -VmName $This.Name -Verbose }
                    }
                }
            }

            Return $Item
        }
        [Object] GetVmDvdDrive()
        {
            $This.Update(0,"[~] Getting VmDvdDrive : $($This.Name)")
            $Item = Switch ($This.Mode)
            {
                Default { Get-VmDvdDrive -VmName $This.Name }
                2       { Get-VmDvdDrive -VmName $This.Name -Verbose }
            }

            Return $Item
        }
        SetVmProcessor()
        {
            $This.Update(0,"[~] Setting VmProcessor (Count): [$($This.Core)]")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Set-VmProcessor -VMName $This.Name -Count $This.Core }
                2       { Set-VmProcessor -VMName $This.Name -Count $This.Core -Verbose }
            }
        }
        SetVmDvdDrive([String]$Path)
        {
            If (![System.IO.File]::Exists($Path))
            {
                $This.Error("[!] Invalid path : [$Path]")
            }

            $This.Update(0,"[~] Setting VmDvdDrive (Path): [$Path]")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Set-VmDvdDrive -VMName $This.Name -Path $Path }
                2       { Set-VmDvdDrive -VMName $This.Name -Path $Path -Verbose }
            }
        }
        SetVmBootOrder([UInt32]$1,[UInt32]$2,[UInt32]$3)
        {
            $This.Update(0,"[~] Setting VmFirmware (Boot order) : [$1,$2,$3]")

            $Fw = $This.GetVmFirmware()

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Set-VMFirmware -VMName $This.Name -BootOrder $Fw.BootOrder[$1,$2,$3] }
                2       { Set-VMFirmware -VMName $This.Name -BootOrder $Fw.BootOrder[$1,$2,$3] -Verbose }
            }
        }
        SetVmSecureBoot([String]$Template)
        {
            $This.Update(0,"[~] Setting VmFirmware (Secure Boot) On, $Template")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default
                {
                    Set-VMFirmware -VMName $This.Name -EnableSecureBoot On -SecureBootTemplate $Template
                }
                2
                {
                    Set-VMFirmware -VMName $This.Name -EnableSecureBoot On -SecureBootTemplate $Template -VB
                }
            }
        }
        AddVmDvdDrive()
        {
            $This.Update(0,"[+] Adding VmDvdDrive")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Add-VmDvdDrive -VMName $This.Name }
                2       { Add-VmDvdDrive -VMName $This.Name -Verbose }
            }
        }
        LoadIso()
        {
            $Item = $This.GetVmDvdDrive()
            If (!$Item.Path -or $Item.Path -ne $This.Image.File.Fullname)
            {
                $This.LoadIso($This.Image.File.Fullname)
            }
        }
        LoadIso([String]$Path)
        {
            If (![System.IO.File]::Exists($Path))
            {
                $This.Error("[!] Invalid ISO path : [$Path]")
            }

            Else
            {
                $This.SetVmDvdDrive($Path)
            }
        }
```

```
        UnloadIso()
        {
            $This.Update(0,"[+] Unloading ISO")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Set-VmDvdDrive -VMName $This.Name -Path $Null }
                2       { Set-VmDvdDrive -VMName $This.Name -Path $Null -Verbose }
            }
        }
        SetIsoBoot()
        {
            If ($This.Generation -eq 2)
            {
                $This.SetVmBootOrder(2,0,1)
            }
        }
        [String[]] GetMacAddress()
        {
            $String = $This.Get().NetworkAdapters[0].MacAddress
            $Mac    = ForEach ($X in 0,2,4,6,8,10)
            {
                $String.Substring($X,2)
            }

            Return $Mac -join "-"
        }
        KeyEntry([Char]$Char)
        {
            $Int = [UInt32]$Char

            If ($Int -in @(33..38+40..43+58+60+62..90+94+95+123..126))
            {
                Switch ($Int)
                {
                    {$_ -in 65..90}
                    {
                        # Lowercase
                        $Int = [UInt32][Char]([String]$Char).ToUpper()
                    }
                    {$_ -in 33,64,35,36,37,38,40,41,94,42}
                    {
                        # Shift+number symbols
                        $Int = Switch ($Int)
                        {
                            33  { 49 } 64  { 50 } 35  { 51 }
                            36  { 52 } 37  { 53 } 94  { 54 }
                            38  { 55 } 42  { 56 } 40  { 57 }
                            41  { 48 }
                        }
                    }
                    {$_ -in 58,43,60,95,62,63,126,123,124,125,34}
                    {
                        # Non-number symbols
                        $Int = Switch ($Int)
                        {
                            58  { 186 } 43  { 187 } 60  { 188 }
                            95  { 189 } 62  { 190 } 63  { 191 }
                            126 { 192 } 123 { 219 } 124 { 220 }
                            125 { 221 } 34  { 222 }
                        }
                    }
                }

                [Void]$This.Keyboard.PressKey(16)
                Start-Sleep -Milliseconds 10

                [Void]$This.Keyboard.TypeKey($Int)
                Start-Sleep -Milliseconds 10

                [Void]$This.Keyboard.ReleaseKey(16)
                Start-Sleep -Milliseconds 10
            }
            Else
            {
                Switch ($Int)
                {
                    {$_ -in 97..122} # Lowercase
                    {
                        $Int = [UInt32][Char]([String]$Char).ToUpper()
                    }
                    {$_ -in 48..57} # Numbers
                    {
                        $Int = [UInt32][Char]$Char
                    }
                    {$_ -in 32,59,61,44,45,46,47,96,91,92,93,39}
                    {
                        $Int = Switch ($Int)
                        {
                            32  {  32 } 59  { 186 } 61  { 187 }
                            44  { 188 } 45  { 189 } 46  { 190 }
                            47  { 191 } 96  { 192 } 91  { 219 }
                            92  { 220 } 93  { 221 } 39  { 222 }
                        }
                    }
                }

                [Void]$This.Keyboard.TypeKey($Int)
                Start-Sleep -Milliseconds 30
            }
        }
        LineEntry([String]$String)
        {
            ForEach ($Char in [Char[]]$String)
            {
                $This.KeyEntry($Char)
            }
        }
        TypeKey([UInt32]$Index)
        {
            $This.Update(0,"[+] Typing key : [$Index]")
            $This.Keyboard.TypeKey($Index)
            Start-Sleep -Milliseconds 125
        }
        PressKey([UInt32]$Index)
        {
            $This.Update(0,"[+] Pressing key : [$Index]")
            $This.Keyboard.PressKey($Index)
        }
        ReleaseKey([UInt32]$Index)
        {
            $This.Update(0,"[+] Releasing key : [$Index]")
            $This.Keyboard.ReleaseKey($Index)
        }
        SpecialKey([UInt32]$Index)
        {
            $This.Keyboard.PressKey(18)
            $This.Keyboard.TypeKey($Index)
            $This.Keyboard.ReleaseKey(18)
        }
        ShiftKey([UInt32[]]$Index)
        {
            $This.Keyboard.PressKey(16)
            ForEach ($X in $Index)
            {
                $This.Keyboard.TypeKey($X)
            }
            $This.Keyboard.ReleaseKey(16)
        }
        TypeCtrlAltDel()
        {
            $This.Update(0,"[+] Typing (CTRL + ALT + DEL)")
            $This.Keyboard.TypeCtrlAltDel()
        }
        TypeChain([UInt32[]]$Array)
        {
            ForEach ($Key in $Array)
            {
                $This.TypeKey($Key)
                Start-Sleep -Milliseconds 125
            }
        }
        TypeLine([String]$String)
        {
            $This.Update(0,"[+] Typing line")
            $This.LineEntry($String)
        }
        TypeText([String]$String)
        {
            $This.Update(0,"[+] Typing text : [$String]")
            $This.LineEntry($String)
        }
        TypeMask([String]$String)
        {
            $This.Update(0,"[+] Typing text : [<Masked>]")
            $This.LineEntry($String)
        }
        TypePassword([Object]$Account)
        {
            $This.Update(0,"[+] Typing password : [<Password>]")
            $This.LineEntry($Account.Password())
            Start-Sleep -Milliseconds 125
        }
        Idle([UInt32]$Percent,[UInt32]$Seconds)
        {
            $This.Update(0,"[~] Idle : $($This.Name) [CPU <= $Percent% for $Seconds second(s)]")

            $C = 0
            Do
            {
                Switch ([UInt32]($This.Get().CpuUsage -le $Percent))
                {
                    0 { $C = 0 } 1 { $C ++ }
                }

                Start-Sleep -Seconds 1
            }
            Until ($C -ge $Seconds)

            $This.Update(1,"[+] Idle complete")
        }
        Uptime([UInt32]$Mode,[UInt32]$Seconds)
        {
            $Mark = @("<=",">=")[$Mode]
            $Flag = 0
            $This.Update(0,"[~] Uptime : $($This.Name) [Uptime $Mark $Seconds second(s)]")
            Do
            {
                Start-Sleep -Seconds 1
                $Uptime        = $This.Get().Uptime.TotalSeconds
                [UInt32] $Flag = Switch ($Mode) { 0 { $Uptime -le $Seconds } 1 { $Uptime -ge $Seconds } }
            }
            Until ($Flag)
            $This.Update(1,"[+] Uptime complete")
        }
        Timer([UInt32]$Seconds)
        {
            $This.Update(0,"[~] Timer : $($This.Name) [Span = $Seconds]")

            $C = 0
            Do
            {
                Start-Sleep -Seconds 1
                $C ++
            }
            Until ($C -ge $Seconds)

            $This.Update(1,"[+] Timer")
        }
        Connection()
        {
            $This.Update(0,"[~] Connection : $($This.Name) [Await response]")

            Do
            {
                Start-Sleep 1
            }
            Until (Test-Connection $This.Network.IpAddress -EA 0)

            $This.Update(1,"[+] Connection")
        }
        [Void] AddScript([UInt32]$Phase,[String]$Name,[String]$DisplayName,[String[]]$Content)
        {
            $This.Script.Add($Phase,$Name,$DisplayName,$Content)
            $This.Update(0,"[+] Added (Script) : $Name")
        }
        [Object] GetScript([UInt32]$Index)
        {
            $Item = $This.Script.Get($Index)
            If (!$Item)
            {
                $This.Error("[!] Invalid index")
            }

            Return $Item
        }
        [Object] GetScript([String]$Name)
        {
            $Item = $This.Script.Get($Name)
            If (!$Item)
            {
                $This.Error("[!] Invalid name")
            }

            Return $Item
        }
        [Void] RunScript()
        {
            $Current = $This.Script.Current()

            If ($Current.Complete -eq 1)
            {
                $This.Error("[!] Exception (Script) : [$($Current.Name)] already completed")
            }

            $This.Update(0,"[~] Running (Script) : [$($Current.Name)]")

            ForEach ($Line in $Current.Content)
            {
                Switch -Regex ($Line)
                {
                    "^\<Idle\[\d+\,\d+\]\>$"
                    {
                        $X = [Regex]::Matches($Line,"\d+").Value
                        $This.Idle($X[0],$X[1])
                    }
                    "^\<Uptime\[\d+\,\d+\]\>$"
                    {
                        $X = [Regex]::Matches($Line,"\d+").Value
                        $This.Uptime($X[0],$X[1])
                    }
                    "^\<Timer\[\d+\]\>$"
                    {
                        $X = [Regex]::Matches($Line,"\d+").Value
                        $This.Timer($X)
                    }
                    "^\<Pass\[.+\]\>$"
                    {
                        $Line = $Matches[0].Substring(6).TrimEnd(">").TrimEnd("]")
                        $This.TypeMask($Line)
                        $This.TypeKey(13)
                    }
                    "^$"
                    {
                        $This.Idle(5,2)
                    }
                    Default
                    {
                        $This.TypeLine($Line)
                        $This.TypeKey(13)
                    }
                }
            }

            $This.Update(1,"[+] Complete (Script) : [$($Current.Name)]")

            $Current.Complete = 1
            $This.Script.Selected ++
        }
        [Void] TransmitScript()
        {
            $Current    = $This.Script.Current()

            If ($Current.Complete -eq 1)
            {
                $This.Error("[!] Exception (Script) : [$($Current.Name)] already completed")
            }

            $This.Update(0,"[~] Transmitting (Script) : [$($Current.Name)]")

            $Content = ForEach ($Line in $Current.Content.Line)
            {
                Switch -Regex ($Line)
                {
                    "^\<Idle\[\d+\,\d+\]\>$"
                    {
                        $Null
                    }
                    "^\<Uptime\[\d+\,\d+\]\>$"
                    {
                        $Null
                    }
                    "^\<Timer\[\d+\]\>$"
                    {
                        $Null
                    }
                    "^\<Pass\[.+\]\>$"
                    {
                        $Null
                    }
                    "^$"
                    {
                        $Null
                    }
                    Default
                    {
                        $Line
                    }
                }
            }

            $Source     = $This.Network.IpAddress
            $Port       = $This.Network.Transmit

            $Command    = @("`$Script = Start-TcpSession -Server -Source $Source -Port $Port",
                            '$Script.Initialize()')

            ForEach ($Item in $Command)
            {
                $This.TypeLine($Item)
                $This.TypeKey(13)
            }

            Start-TcpSession -Client -Source $Source -Port $Port -Content $Content | % Initialize

            $This.TypeLine('$Script.Content.Message -join "" | Invoke-Expression')
            $This.TypeKey(13)

            $This.Update(1,"[+] Complete (Script) : [$($Current.Name)]")

            $Current.Complete     ++
            $This.Script.Selected ++
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Object]>"
        }
    }
```

```
PS Prompt:\> $Vm

Console    : 00:06:10.3713525
Name       : desktop01
Role       : Client
Memory     : 4.00 GB
Path       : C:\VDI\desktop01\desktop01
Vhd        : C:\VDI\desktop01\desktop01.vhdx
VhdSize    : 68719476736
Generation : 2
Core       : 2
Switch     : {External}
Firmware   :
Exists     : 1
Guid       : d6c5d9df-b0b8-4498-b8c0-5815824e8c1a
Account    : {<FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>}
Network    : <FEVirtual.VmNode[Network]>
Image      : <FEVirtual.VmNodeImage[Object]>
Script     : <FEVirtual.VmNodeScriptBlock[Controller]>
Checkpoint : {<FEVirtual.VmCheckpoint>, <FEVirtual.VmCheckpoint>}

PS Prompt:\>
```

```
                                                                                                 ______________________/
\_______________________________________________________________________________________________/ Class [VmNodeObject]
  Class [VmNodeWindows] /----------------------------------------------------------------------------------------------\
/-----------------------
```

    (...some text wrapping...) This class is actually an [extension] of the [above class].

```
    Class VmNodeWindows : VmNodeObject
    {
        VmNodeWindows([Switch]$Flags,[Object]$Vm) : base($Flags,$Vm)
        {

        }
        VmNodeWindows([Object]$File) : base($File)
        {

        }
        [UInt32] NetworkSetupMode()
        {
            $Arp = (arp -a) -match $This.GetMacAddress() -Split " " | ? Length -gt 0

            Return !!$Arp
        }
        SetAdmin([Object]$Account)
        {
            $This.Update(0,"[~] Setting : Administrator password")
            ForEach ($X in 0..1)
            {
                $This.TypePassword($Account)
                $This.TypeKey(9)
                Start-Sleep -Milliseconds 125
            }

            $This.TypeKey(9)
            Start-Sleep -Milliseconds 125
            $This.TypeKey(13)
        }
        Login([Object]$Account)
        {
            $This.Update(0,"[~] Login : [Account: $($Account.Username)")
            $This.TypeCtrlAltDel()
            $This.Timer(5)
            $This.TypePassword($Account)
            Start-Sleep -Milliseconds 125
            $This.TypeKey(13)
        }
        LaunchPs()
        {
            # Open Start Menu
            $This.PressKey(91)
            $This.TypeKey(88)
            $This.ReleaseKey(91)
            $This.Timer(2)

            Switch ($This.Role)
            {
                Server
                {
                    # Open Command Prompt
                    $This.TypeKey(65)
                    $This.Timer(2)

                    # Maximize window
                    $This.PressKey(91)
                    $This.TypeKey(38)
                    $This.ReleaseKey(91)
                    $This.Timer(1)

                    # Start PowerShell
                    $This.TypeText("PowerShell")
                    $This.TypeKey(13)
                    $This.Timer(1)
                }
                Client
                {
                    # // Open [PowerShell]
                    $This.TypeKey(65)
                    $This.Timer(2)
                    $This.TypeKey(37)
                    $This.Timer(2)
                    $This.TypeKey(13)
                    $This.Timer(4)

                    # // Maximize window
                    $This.PressKey(91)
                    $This.TypeKey(38)
                    $This.ReleaseKey(91)
                    $This.Timer(1)
                }
            }

            # Wait for PowerShell engine to get ready for input
            $This.Idle(5,5)
        }
        [String[]] Initialize()
        {
```

```
        # Set IP Address
        $Content = @(
        '$Index = Get-NetAdapter | ? Status -eq Up | % InterfaceIndex';
        '$Interface = Get-NetIPAddress -AddressFamily IPv4 -InterfaceIndex $Index';
        '$Interface | Remove-NetIPAddress -AddressFamily IPv4 -Confirm:0 -Verbose';
        '$Interface | Remove-NetRoute -AddressFamily IPv4 -Confirm:0 -Verbose';
        '$Splat = @{';
        '    InterfaceIndex  = $Index';
        '    AddressFamily = "IPv4"';
        '    PrefixLength = {0}' -f $This.Network.Prefix;
        '    ValidLifetime = [Timespan]::MaxValue';
        '    IPAddress = "{0}"' -f $This.Network.IpAddress;
        '    DefaultGateway = "{0}"' -f $This.Network.Gateway;
        '}';
        'New-NetIPAddress @Splat';
        'Set-DnsClientServerAddress -InterfaceIndex $Index -ServerAddresses {0} -Verbose' -f
($This.Network.Dns -join ',');
        "`$Desc = 'Allows content to be {0} over TCP/$($This.Network.Transmit)'";
        '$Splat = @{ ';
        '    Description = $Desc -f "sent"';
        '    LocalPort = {0}' -f $This.Network.Transmit;
        '}';
        'New-NetFirewallRule @Splat -Direction Inbound -DisplayName TCPSession -Protocol TCP -Action
Allow -Verbose';
        '$Splat = @{';
        '    Description = $Desc -f "received"';
        '    RemotePort  = {0}' -f $This.Network.Transmit;
        '}';
        'New-NetFirewallRule @Splat -Direction Outbound -DisplayName TCPSession -Protocol TCP -Action
Allow -Verbose';
        '$Base = "https://www.github.com/mcc85s/FightingEntropy/blob/main/Version/2023.4.0"'
        '$Url = "$Base/FightingEntropy.ps1?raw=true"';
        'Invoke-RestMethod $Url | Invoke-Expression';
        '$Module.Latest()')

        Return $Content
    }
    [String[]] ImportFeModule()
    {
        Return 'Set-ExecutionPolicy Bypass -Scope Process -Force', 'Import-Module FightingEntropy -Force
-Verbose'
    }
    [String[]] PrepPersistentInfo()
    {
        # Prepare the correct persistent information
        $List = @( )

        $List += '$P = @{ }'
        ForEach ($P in @($This.Network.PSObject.Properties | ? Name -ne Dhcp))
        {
            $List += Switch -Regex ($P.TypeNameOfValue)
            {
                Default
                {
                    '$P.Add($P.Count,("{0}","{1}"))' -f $P.Name, $P.Value
                }
                "\[\]"
                {
                    '$P.Add($P.Count,("{0}",@([String[]]"{1}")))' -f $P.Name, ($P.Value -join "`",`"")
                }
            }
        }

        If ($This.Role -eq "Server")
        {
            $List += '$P.Add($P.Count,("Dhcp","$Dhcp"))'
        }

        $List += '$P[0..($P.Count-1)] | % { Set-ItemProperty -Path $Path -Name $_[0] -Value $_[1]
-Verbose }'

        If ($This.Role -eq "Server")
        {
            $List += '$P = @{ }'

            ForEach ($P in @($This.Network.Dhcp.PSObject.Properties))
            {
                $List += Switch -Regex ($P.TypeNameOfValue)
                {
                    Default
                    {
                        '$P.Add($P.Count,("{0}","{1}"))' -f $P.Name, $P.Value
                    }
                    "\[\]"
                    {
                        '$P.Add($P.Count,("{0}",@([String[]]"{1}")))' -f $P.Name, ($P.Value -join
"`",`"")
                    }
                }
            }

            $List += '$P[0..($P.Count-1)] | % { Set-ItemProperty -Path $Dhcp -Name $_[0] -Value $_[1]
-Verbose }'
        }

        Return $List
    }
    SetPersistentInfo()
    {
        # [Phase 1] Set persistent information
        $This.Script.Add(1,"SetPersistentInfo","Set persistent information",@(
        '$Root      = "{0}"' -f $This.GetRegistryPath();
        '$Name      = "{0}"' -f $This.Name;
        '$Path      = "$Root\ComputerInfo"';
        'Rename-Computer $Name -Force -EA 0';
        'If (!(Test-Path $Root))';
        '{';
        '    New-Item -Path $Root -Verbose';
        '}';
        'New-Item -Path $Path -Verbose';
        If ($This.Role -eq "Server")
        {
            '$Dhcp = "$Path\Dhcp"';
            'New-Item $Dhcp';
        }
        $This.PrepPersistentInfo()))
    }
    SetTimeZone()
    {
        # [Phase 2] Set time zone
        $This.Script.Add(2,"SetTimeZone","Set time zone",@('Set-Timezone -Name "{0}" -Verbose' -f (Get-
Timezone).Id))
    }
    SetComputerInfo()
    {
        # [Phase 3] Set computer info
        $This.Script.Add(3,"SetComputerInfo","Set computer info",@(
        '$Item           = Get-ItemProperty "{0}\ComputerInfo"' -f $This.GetRegistryPath()
        '$TrustedHost    = $Item.Trusted';
        '$IPAddress      = $Item.IpAddress';
        '$PrefixLength   = $Item.Prefix';
        '$DefaultGateway = $Item.Gateway';
        '$Dns            = $Item.Dns'))
    }
    SetIcmpFirewall()
    {
        $Content = Switch ($This.Role)
        {
            Server
            {
                'Get-NetFirewallRule | ? DisplayName -match "(Printer.+IcmpV4)" | Enable-NetFirewallRule
-Verbose'
            }
            Client
            {
                'Get-NetFirewallRule | ? DisplayName -match "(Printer.+IcmpV4)" | Enable-NetFirewallRule
-Verbose',
                'Get-NetConnectionProfile | Set-NetConnectionProfile -NetworkCategory Private -Verbose'
            }
        }

        # [Phase 4] Enable IcmpV4
        $This.Script.Add(4,"SetIcmpFirewall","Enable IcmpV4",@($Content))
    }
    SetInterfaceNull()
    {
        # [Phase 5] Get InterfaceIndex, get/remove current (IP address + Net Route)
        $This.Script.Add(5,"SetInterfaceNull","Get InterfaceIndex, get/remove current (IP address + Net
Route)",@(
        '$Index              = Get-NetAdapter | ? Status -eq Up | % InterfaceIndex';
        '$Interface          = Get-NetIPAddress    -AddressFamily IPv4 -InterfaceIndex $Index';
        '$Interface          | Remove-NetIPAddress -AddressFamily IPv4 -Confirm:$False -Verbose';
        '$Interface          | Remove-NetRoute     -AddressFamily IPv4 -Confirm:$False -Verbose'))
    }
    SetStaticIp()
    {
        # [Phase 6] Set static IP Address
        $This.Script.Add(6,"SetStaticIp","Set (static IP Address + Dns server)",@(
        '$Splat              = @{';
        ' ';
        '    InterfaceIndex  = $Index';
        '    AddressFamily   = "IPv4"';
        '    PrefixLength    = $Item.Prefix';
        '    ValidLifetime   = [Timespan]::MaxValue';
        '    IPAddress       = $Item.IPAddress';
        '    DefaultGateway  = $Item.Gateway';
        '}';
        'New-NetIPAddress @Splat';
        'Set-DnsClientServerAddress -InterfaceIndex $Index -ServerAddresses $Item.Dns'))
    }
    SetWinRm()
    {
        # [Phase 7] Set WinRM (Config)
        $This.Script.Add(7,"SetWinRm","Set (WinRM Config/Self-Signed Certificate/HTTPS Listener)",@(
        'winrm quickconfig';
        '<Timer[2]>';
        'y';
        '<Timer[3]>';
        If ($This.Role -eq "Client")
        {
            'y';
            '<Timer[3]>';
        }
        'Set-Item WSMan:\localhost\Client\TrustedHosts -Value $Item.Trusted';
        '<Timer[4]>';
        'y'))
    }
    SetWinRmFirewall()
    {
        # [Phase 8] Set WinRm (Self-Signed Certificate/HTTPS Listener/Firewall)
        $This.Script.Add(8,"SetWinRmFirewall",'Set WinRm Firewall',@(
        '$Cert           = New-SelfSignedCertificate -DnsName $Item.IpAddress -CertStoreLocation
Cert:\LocalMachine\My';
        '$Thumbprint     = $Cert.Thumbprint';
        '$Hash           = "@{Hostname=`"$IPAddress`";CertificateThumbprint=`"$Thumbprint`"}"';
        "`$Str            = `"winrm create winrm/config/Listener?Address=*+Transport=HTTPS '{0}'`"";
        'Invoke-Expression ($Str -f $Hash)'
        '$Splat          = @{';
        ' ';
        '    Name        = "WinRM/HTTPS"';
        '    DisplayName = "Windows Remote Management (HTTPS-In)"';
        '    Direction   = "In"';
        '    Action      = "Allow"';
        '    Protocol    = "TCP"';
        '    LocalPort   = 5986';
        '}';
        'New-NetFirewallRule @Splat -Verbose'))
    }
    SetRemoteDesktop()
    {
        # [Phase 9] Set Remote Desktop
        $This.Script.Add(9,"SetRemoteDesktop",'Set Remote Desktop',@(
        'Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\Terminal Server" -Name
fDenyTSConnections -Value 0';
        'Enable-NetFirewallRule -DisplayGroup "Remote Desktop"'))
    }
    InstallFeModule()
    {
        # [Phase 10] Install [FightingEntropy()]
        $This.Script.Add(10,"InstallFeModule","Install [FightingEntropy()]",@(
        '[Net.ServicePointManager]::SecurityProtocol = 3072';
        'Set-ExecutionPolicy Bypass -Scope Process -Force';
        '$Install =
"https://github.com/mcc85s/FightingEntropy/blob/main/Version/2023.4.0/FightingEntropy.ps1?raw=true"';
        'Invoke-RestMethod $Install | Invoke-Expression';
        '$Module.Latest()';
        '<Idle[5,5]>';
        'Import-Module FightingEntropy'))
    }
    InstallChoco()
    {
        # [Phase 11] Install Chocolatey
        $This.Script.Add(11,"InstallChoco","Install Chocolatey",@(
        "Invoke-RestMethod https://chocolatey.org/install.ps1 | Invoke-Expression"))
    }
    InstallVsCode()
    {
        # [Phase 12] Install Visual Studio Code
        $This.Script.Add(12,"InstallVsCode","Install Visual Studio Code",@("choco install vscode -y"))
    }
    InstallBossMode()
    {
        # [Phase 13] Install BossMode (vscode color theme)
        $This.Script.Add(13,"InstallBossMode","Install BossMode (vscode color theme)",@("Install-
BossMode"))
    }
    InstallPsExtension()
    {
        # [Phase 14] Install Visual Studio Code (PowerShell Extension)
        $This.Script.Add(14,"InstallPsExtension","Install Visual Studio Code (PowerShell Extension)",@(
        '$FilePath     = "$Env:ProgramFiles\Microsoft VS Code\bin\code.cmd"';
        '$ArgumentList = "--install-extension ms-vscode.PowerShell"';
        'Start-Process -FilePath $FilePath -ArgumentList $ArgumentList -NoNewWindow | Wait-Process'))
    }
    RestartComputer()
    {
        # [Phase 15] Restart computer
        $This.Script.Add(15,'Restart','Restart computer',@('Restart-Computer'))
    }
    ConfigureDhcp()
    {
        # [Phase 16] Configure Dhcp
        $This.Script.Add(16,'ConfigureDhcp','Configure Dhcp',@(
        '$Root           = "{0}"' -f $This.GetRegistryPath()
        '$Path           = "$Root\ComputerInfo"'
        '$Item           = Get-ItemProperty $Path'
        '$Item.Dhcp      = Get-ItemProperty $Item.Dhcp';
        ' ';
        '$Splat = @{ ';
        '   ';
        '    StartRange = $Item.Dhcp.StartRange';
        '    EndRange   = $Item.Dhcp.EndRange';
        '    Name       = $Item.Dhcp.Name';
        '    SubnetMask = $Item.Dhcp.SubnetMask';
        '}';
        ' ';
        'Add-DhcpServerV4Scope @Splat -Verbose';
        'Add-DhcpServerInDc -Verbose';
        ' ';
        'ForEach ($Value in $Item.Dhcp.Exclusion)';
        '{';
        '    $Splat         = @{ ';
        ' ';
        '        ScopeId    = $Item.Dhcp.Network';
        '        StartRange = $Value';
        '        EndRange   = $Value';
        '    }';
        ' ';
        '    Add-DhcpServerV4ExclusionRange @Splat -Verbose';
        ' ';
        '   (3,$Item.Gateway),';
        '   (6,$Item.Dns),';
        '   (15,$Item.Domain),';
        '   (28,$Item.Dhcp.Broadcast) | % {';
        '     ';
        '       Set-DhcpServerV4OptionValue -OptionId $_[0] -Value $_[1] -Verbose'
        '   }';
        '}';
        'netsh dhcp add securitygroups';
        'Restart-Service dhcpserver';
        ' ';
        '$Splat    = @{ ';
        ' ';
        '    Path  = "HKLM:\SOFTWARE\Microsoft\ServerManager\Roles\12"';
        '    Name  = "ConfigurationState"';
        '    Value = 2';
        '}';
        ' ';
        'Set-ItemProperty @Splat -Verbose'))
    }
    InitializeFeAd([String]$Pass)
    {
        $This.Script.Add(17,'InitializeAd','Initialize [FightingEntropy()] AdInstance',@(
        '$Password = Read-Host "Enter password" -AsSecureString';
        '<Timer[2]>';
        '{0}' -f $Pass;
        '$Ctrl = Initialize-FeAdInstance';
        ' ';
        '# Set location';
        '$Ctrl.SetLocation("1718 US-9","Clifton Park","NY",12065,"US")';
        ' ';
        '# Add Organizational Unit';
        '$Ctrl.AddAdOrganizationalUnit("DevOps","Developer(s)/Operator(s)")';
        ' ';
        '# Get Organizational Unit';
        '$Ou     = $Ctrl.GetAdOrganizationalUnit("DevOps")';
        ' ';
        '# Add Group';
        '$Ctrl.AddAdGroup("Engineering","Security","Global","Secure Digits Plus LLC",
$Ou.DistinguishedName)';
        ' ';
        '# Get Group';
        '$Group  = $Ctrl.GetAdGroup("Engineering")';
        ' ';
        '# Add-AdPrincipalGroupMembership';
        '$Ctrl.AddAdPrincipalGroupMembership($Group.Name,@("Administrators","Domain Admins"))';
        ' ';
        '# Add User';
        '$Ctrl.AddAdUser("Michael","C","Cook","mcook85",$Ou.DistinguishedName)';
        ' ';
        '# Get User';
        '$User   = $Ctrl.GetAdUser("Michael","C","Cook")';
        ' ';
        '# Set [User.General (Description, Office, Email, Homepage)]';
        '$User.SetGeneral("Beginning the fight against ID theft and cybercrime",';
        '                 "<Unspecified>",';
        '                 "michael.c.cook.85@gmail.com",';
        '                 "https://github.com/mcc85s/FightingEntropy")';
        ' ';
        '# Set [User.Address (StreetAddress, City, State, PostalCode, Country)] ';
```

```
            '$User.SetLocation($Ctrl.Location)';
            ' ';
            '# Set [User.Profile (ProfilePath, ScriptPath, HomeDirectory, HomeDrive)]';
            '$User.SetProfile("","","","")';
            ' ';
            '# Set [User.Telephone (HomePhone, OfficePhone, MobilePhone, Fax)]';
            '$User.SetTelephone("","518-406-8569","518-406-8569","")';
            ' ';
            '# Set [User.Organization (Title, Department, Company)]';
            '$User.SetOrganization("CEO/Security Engineer","Engineering","Secure Digits Plus LLC")';
            ' ';
            '# Set [User.AccountPassword]';
            '$User.SetAccountPassword($Password)';
            ' ';
            '# Add user to group';
            '$Ctrl.AddAdGroupMember($Group,$User)';
            ' ';
            '# Set user primary group';
            '$User.SetPrimaryGroup($Group)'))
        }
        Load()
        {
            $This.SetPersistentInfo()
            $This.SetTimeZone()
            $This.SetComputerInfo()
            $This.SetIcmpFirewall()
            $This.SetInterfaceNull()
            $This.SetStaticIp()
            $This.SetWinRm()
            $This.SetWinRmFirewall()
            $This.SetRemoteDesktop()
            $This.InstallFeModule()
            $This.InstallChoco()
            $This.InstallVsCode()
            $This.InstallBossMode()
            $This.InstallPsExtension()
            $This.RestartComputer()
            $This.ConfigureDhcp()
        }
        [Object] PSSession([Object]$Account)
        {
            # Creates session object
            $This.Update(0,"[~] PSSession Token")
            $Splat = @{

                ComputerName  = $This.Network.IpAddress
                Port          = 5986
                Credential    = $Account.Credential
                SessionOption = New-PSSessionOption -SkipCACheck
                UseSSL        = $True
            }

            Return $Splat
        }
    }
```

```
                                                                                                ________________________/
\_____________________________________________________________________________________________/ Class [VmNodeWindows]
  Class [VmNodeLinux] /-----------------------------------------------------------------------------------------------\
/---------------------
```

```
(...some text wrapping...) This class is actually an extension of the above class [VmNodeObject]
```

```
    Class VmNodeLinux : VmNodeObject
    {
        VmNodeLinux([Switch]$Flags,[Object]$Vm) : base($Flags,$Vm)
        {
        }
        VmNodeLinux([Object]$File) : base($File)
        {

        }
        Login([Object]$Account)
        {
            # Login
            $This.Update(0,"Login [+] [$($This.Name): $([DateTime]::Now)]")
            $This.TypeKey(9)
            $This.TypeKey(13)
            $This.Timer(1)
            $This.TypePassword($Account.Password())
            $This.TypeKey(13)
            $This.Idle(0,5)
        }
        Initial()
        {
            $This.Update(0,"Running [~] Initial Login")
            # Learn your way around...?

            $This.TypeKey(32)
            $This.Timer(1)
            $This.TypeKey(27)
            $This.Timer(1)
        }
        LaunchTerminal()
        {
            $This.Update(0,"Launching [~] Terminal")

            # // Launch terminal
            $This.TypeKey(91)
            $This.Timer(2)
            $This.TypeLine("terminal")
            $This.Timer(2)
            $This.TypeKey(13)
            $This.Timer(2)

            # // Maximize window
            $This.PressKey(91)
            $This.TypeKey(38)
            $This.ReleaseKey(91)
            $This.Idle(0,5)
        }
        Super([Object]$Account)
        {
            $This.Update(0,"Super User [~]")

            # // Accessing super user
            ForEach ($Key in [Char[]]"su -")
            {
                $This.LinuxKey($Key)
                Start-Sleep -Milliseconds 25
            }

            $This.TypeKey(13)
            $This.Timer(1)
            $This.LinuxPassword($Account.Password())
            $This.TypeKey(13)
            $This.Idle(5,2)
        }
        [String] RichFirewallRule()
        {
            $Line = "firewall-cmd --permanent --zone=public --add-rich-rule='"
            $Line += 'rule family="ipv4" '
            $Line += 'source address="{0}/{1}" ' -f $This.Network.Ipaddress, $This.Network.Prefix
            $Line += 'port port="3389" '
            $Line += "protocol=`"tcp`" accept'"

            Return $Line
        }
        SubscriptionInfo([Object]$User)
        {
            # [Phase 1] Set subscription service to access (yum/rpm)
            $This.Script.Add(1,"SetSubscriptionInfo","Set subscription information",@(
            "subscription-manager register";
            "<Timer[1]>";
            $User.Username;
            "<Timer[1]>";
            "<Pass[$($User.Password())]>";
            ))
        }
        GroupInstall()
        {
            # [Phase 2] Install groupinstall workgroup
            $This.Script.Add(2,"GroupInstall","Install groupinstall workgroup",@(
            "dnf groupinstall workstation -y";
            "";
            ))
        }
        InstallEpel()
        {
            # [Phase 3] (Set/Install) epel-release
            $This.Script.Add(3,"EpelRelease","Set EPEL Release Repo",@(
            'subscription-manager repos --enable codeready-builder-for-rhel-9-x86_64-rpms';
            "<Timer[30]>";
            "";
            "dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm -y";
            "";
            ))
        }
        InstallPs()
        {
            # [Phase 4] (Set/Install) [PowerShell]
            $This.Script.Add(4,"InstallPs","(Set/Install) [PowerShell]",@(
            "curl https://packages.microsoft.com/config/rhel/8/prod.repo | tee
    /etc/yum.repos.d/microsoft.repo";
            "";
            "dnf install powershell -y"
            ))
        }
        InstallRdp()
        {
            # [Phase 5] Install [Remote Desktop] Tools
            $This.Script.Add(5,"InstallRdp","(Set/Install) [Remote Desktop] Tools",@(
            "dnf install tigervnc-server tigervnc -y";
            "<Timer[5]>";
            "";
            "yum --enablerepo=epel install xrdp -y";
            "<Timer[5]>";
            "";
            "systemctl start xrdp.service";
            "";
            "systemctl enable xrdp.service"
            ""
            ))
        }
        SetFirewall()
        {
            # [Phase 6] Set firewall
            $This.Script.Add(6,"SetFirewall","Set firewall rule and restart",@(
            $This.RichFirewallRule();
            "";
            "firewall-cmd --reload"
            ))
        }
        InstallVSCode()
        {
            # [Phase 7] Install [Visual Studio Code]
            $This.Script.Add(7,"InstallVsCode","(Set/Install) [Visual Studio Code]",@(
            '$Link  = "https://packages.microsoft.com"';
            '$Keys  = "{0}/keys/microsoft.asc" -f $Link';
            '$Repo  = "{0}/yumrepos/vscode" -f $Link';
            '$Path  = "/etc/yum.repos.d/vscode.repo"';
```

```
            '$Text  = @( )';
            '$Text += "[code]"';
            '$Text += "name=Visual Studio Code"';
            '$Text += "baseurl={0}" -f $Repo';
            '$Text += "enabled=1"';
            '$Text += "gpgcheck=1"';
            '$Text += "gpgkey={0}" -f $Keys';
            '[System.IO.File]::WriteAllLines($Path,$Text)';
            "";
            'rpm --import $Keys';
            "";
            'yum install code -y'
            ))
        }
        InstallPsExtension()
        {
            # [Phase 8] Install [PowerShell Extension]
            $This.Script.Add(7,"InstallPsExtension","Install [PowerShell Extension]",@(
            'code --install-extension ms-vscode.powershell'
            ))
        }
        Load([Object]$User)
        {
            $This.SubscriptionInfo($User)
            $This.GroupInstall()
            $This.InstallEpel()
            $This.InstallPs()
            $This.InstallRdp()
            $This.SetFirewall()
            $This.InstallVSCode()
            $This.InstallPsExtension()
        }
    }
```

```
                                                                                                   _____________________/
\__________________________________________________________________________________________________/ Class [VmNodeLinux]
  Class [VmNodeMaster] /----------------------------------------------------------------------------------------------\
/---------------------
```

So, all of the above [node classes] are controlled by the [node master].

This object is a subordinate class of the [VmControllerMaster] object which combines the XAML and validation flags, among many other things related to borrowing elements from all of the above classes.

```
    Class VmNodeMaster
    {
        [UInt32] $Selected
        [String]     $Path
        [Object]   $Switch
        [Object]     $Host
        [Object] $Template
        [Object]   $Object
        VmNodeMaster()
        {
            $This.Refresh()
        }
        SetPath([String]$Path)
        {
            If (![System.IO.Directory]::Exists($Path))
            {
                Throw "Invalid path"
            }

            $This.Path = $Path
        }
        Select([UInt32]$Index)
        {
            If ($Index -gt $This.Object.Count)
            {
                Throw "Invalid index"
            }

            $This.Selected = $Index
        }
        [Object] Current()
        {
            Return $This.Object[$This.Selected]
        }
        Clear([String]$Slot)
        {
            Switch -Regex ($Slot)
            {
                "Switch"   { $This.Switch   = @( ) }
                "Host"     { $This.Host     = @( ) }
                "Template" { $This.Template = @( ) }
                "Object"   { $This.Object   = @( ) }
            }
        }
        [Object] VmNodeSwitch([UInt32]$Index,[Object]$VmSwitch)
        {
            Return [VmNodeSwitch]::New($Index,$VmSwitch)
        }
        [Object] VmNodeHost([UInt32]$Index,[Object]$VmNode)
        {
            Return [VmNodeHost]::New($Index,$VmNode)
        }
        [Object] VmNodeTemplate([UInt32]$Index,[Object]$File)
        {
            Return [VmNodeTemplate]::New($Index,$File)
        }
        [Object] VmNodeSlot([UInt32]$Index,[Object]$Node)
        {
            Return [VmNodeSlot]::New($Index,$Node)
        }
        [Object] VmNodeObject([Object]$Node)
        {
            Return [VmNodeObject]::New($Node)
        }
        [Object] VmNodeWindows([Object]$Node)
        {
            Return [VmNodeWindows]::New($Node)
        }
        [Object] VmNodeLinux([Object]$Node)
        {
            Return [VmNodeLinux]::New($Node)
        }
        [Object[]] GetVmSwitch()
        {
            Return Get-VmSwitch
        }
        [Object[]] GetVm()
        {
            Return Get-Vm
        }
        [Object[]] GetTemplate()
        {
            Return Get-ChildItem $This.Path | ? Extension -eq .fex
        }
        NewVmSwitch([String]$Name,[String]$Type)
        {
            New-VmSwitch -Name $Name -SwitchType $Type -Verbose
            $This.Refresh("Switch")
        }
        RemoveVmSwitch([String]$Name)
        {
            Remove-VmSwitch -Name $Name -Force -Verbose
            $This.Refresh("Switch")
        }
        [Object] Create([UInt32]$Index)
        {
            If (!$This.Template[$Index])
            {
                Throw "Invalid index"
            }

            If ($This.Template[$Index].Name -in $This.Object)
            {
                Throw "Item is already in the object list"
            }

            $Temp = $This.Template[$Index]
            $Item = Switch -Regex ($Temp.Role)
            {
                "(^Server$|^Client$)"
                {
                    $This.VmNodeWindows($Temp)
                }
                "(^Linux$)"
                {
                    $This.VmNodeLinux($Temp)
                }
            }

            Return $Item
        }
        AddTemplate([Object]$Template)
        {
            $This.Template += $This.VmNodeTemplate($This.Template.Count,$Template)
        }
        AddSwitch([Object]$VmSwitch)
        {
            $This.Switch   += $This.VmNodeSwitch($This.Switch.Count,$VmSwitch)
        }
        AddHost([Object]$Node)
        {
            $This.Host     += $This.VmNodeHost($This.Host.Count,$Node)
        }
        AddObject([Object]$Node)
        {
            $This.Object   += $This.VmNodeSlot($This.Object.Count,$Node)
        }
        Refresh([String]$Type)
        {
            If ($Type -notin "Switch","Host","Template","Object")
            {
                Throw "Invalid type"
            }

            $This.Clear($Type)

            Switch ($Type)
            {
                "Switch"
                {
                    ForEach ($Item in $This.GetVmSwitch())
                    {
                        $This.AddSwitch($Item)
                    }
                }
                "Host"
                {
                    ForEach ($Item in $This.GetVm())
                    {
                        $This.AddHost($Item)
                    }
                }
                "Template"
                {
                    If ($This.Path)
                    {
                        ForEach ($Item in $This.GetTemplate())
                        {
                            $This.AddTemplate($Item)
                        }
                    }
                }
                "Object"
                {
                    ForEach ($Item in $This.Host)
                    {
                        $This.Object += $This.VmNodeSlot($This.Object.Count,$Item)
                    }

                    ForEach ($Item in $This.Template)
                    {
                        $This.Object += $This.VmNodeSlot($This.Object.Count,$Item)
                    }
                }
            }
        }
        Refresh()
        {
            ForEach ($Item in "Switch","Host","Template","Object")
            {
                $This.Refresh($Item)
            }
        }
        [Object] Control([String]$Path)
        {
            If (![System.IO.File]::Exists($Path))
            {
                Throw "Invalid path"
            }

            Return $This.VmNodeWindows($This.VmNodeTemplate(0,(Get-Item $Path)))
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Master]>"
        }
    }
```

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

Selected : 0
Path     : C:\FileVm
Switch   : {<FEVirtual.VmNode[Switch]>, <FEVirtual.VmNode[Switch]>}
Host     : {<FEVirtual.VmNode[Host]>}
Template : {<FEVirtual.VmNode[Template]>}
Object   : {desktop01, desktop01}

PS Prompt:\>
```

```
                                                                                                    ______________________/
\__________________________________________________________________________________________/ Class [VmNodeMaster]
  Class [VmControllerProperty] /------------------------------------------------------------------------------------\
/------------------------------
```

This allows the GUI to be able to divide [property (names/values)].

```

    Class VmControllerProperty
    {
        [String]  $Name
        [Object] $Value
        VmControllerProperty([Object]$Property)
        {
            $This.Name  = $Property.Name
            $This.Value = $Property.Value -join ", "
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmController[Property]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Xaml.IO.NodeHostExtension.Items

Name      Value
----      -----
Index     0
Guid      705cdce0-3c62-492b-9b2f-659e5c7166c0
Name      desktop01
Role      Client
Account   <FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>, <FEVirtual.VmCredential[Item]>
IPAddress 192.168.42.1
Domain    securedigitsplus.com
NetBios   SECURED
Trusted   192.168.42.2
Prefix    24
Netmask   255.255.255.0
Gateway   192.168.42.129
Dns       192.168.42.129
Dhcp      <FEVirtual.VmNode[Dhcp]>
Base      C:\VDI
Memory    4294967296
Hdd       68719476736
Gen       2
Core      2
SwitchId  External
Image     <FEVirtual.VmNodeImage[Object]>


PS Prompt:\>
```

```
                                                                                         ______________________________/
\_______________________________________________________________________________________/ Class [VmControllerProperty]
  Class [VmControllerFlag] /‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

This allows a [Xaml control] to have an [attributable (name/status)] for [validation].

[Validation] is used to control the status of the icons and [enabling/disabling] various components of the GUI.

```
    Class VmControllerFlag
    {
        [UInt32] $Index
        [String] $Name
        [UInt32] $Status
        VmControllerFlag([UInt32]$Index,[String]$Name)
        {
            $This.Index  = $Index
            $This.Name   = $Name
            $This.SetStatus(0)
        }
        SetStatus([UInt32]$Status)
        {
            $This.Status = $Status
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmController[Flag]>"
        }
    }
```

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

```
Index Name                 Status
----- ----                 ------
    0 MasterPath                1
    1 MasterDomain              1
    2 MasterNetBios             1
    3 CredentialUsername        0
    4 CredentialPassword        0
    5 CredentialConfirm         0
    6 CredentialPin             0
    7 ImagePath                 1
    8 TemplateName              1
    9 TemplatePath              1
   10 TemplateImagePath         1
   11 NodeTemplatePath          1


PS Prompt:\>
```

```
                                                                                                         ___________________________/
\__________________________________________________________________________________________________________/ Class [VmControllerFlag]
  Class [VmControllerCredential] /-----------------------------------------------------------------------------------------------------\
/--------------------------------
```

Converts a [credential object] into a [Xaml DataGrid] object.
It adds the ability to select an empty object so that a [new credential object] can be [tested] and [validated].
It also allows the [removal] of those objects. (*Exporting not quite ready*)

```
    Class VmControllerCredential
    {
        [String]    $Index
        [Guid]       $Guid
        [String]     $Type
        [String] $Username
        [String]     $Pass
        VmControllerCredential([Object]$Account)
        {
            $This.Index    = $Account.Index
            $This.Guid     = $Account.Guid
            $This.Type     = $Account.Type
            $This.Username = $Account.Username
            $This.Pass     = $Account.Pass
        }
        VmControllerCredential()
        {
            $This.Guid     = $This.NewGuid()
            $This.Type     = "<New>"
        }
        [Object] NewGuid()
        {
            Return [Guid]::NewGuid()
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmController[Credential]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Xaml.IO.CredentialOutput.Items | Format-Table

Index Guid                                 Type      Username           Pass
----- ----                                 ----      --------           ----
0     0c98ed6c-7a92-4dd4-bb05-b648387984f2 Setup     Administrator      <SecureString>
1     cd6c175d-2869-4a95-9e7c-81eef6cb43d1 User      mcc85s             <SecureString>
2     66ebdf6f-520f-4a22-9247-7eaa47f96928 Microsoft sdp12065@gmail.com <SecureString>
      1e3300b3-748b-45bd-afaf-9236d3d5beaf <New>

PS Prompt:\>
```

```
                                                                                   ________________________________/
\_________________________________________________________________________________/ Class [VmControllerCredential]
  Class [VmControllerTemplate] /--------------------------------------------------------------------------------------\
/------------------------------
```

Same idea as the above, strictly meant to allow [templates] to be [created], [amended], or [removed].

```
    Class VmControllerTemplate
    {
        [String]    $Index
        [Guid]       $Guid
        [String]     $Name
        [String]     $Role
        [String]     $Base
        [String]   $Memory
        [String]      $Hdd
        [String]      $Gen
        [String]     $Core
        [String] $SwitchId
        [String]    $Image
        VmControllerTemplate([Object]$Object)
        {
            $This.Index    = $Object.Index
            $This.Guid     = $Object.Guid
            $This.Name     = $Object.Name
            $This.Role     = $Object.Role
            $This.Base     = $Object.Path
            $This.Memory   = $Object.Ram
            $This.Hdd      = $Object.Hdd
            $This.Gen      = $Object.Gen
            $This.Core     = $Object.Core
            $This.SwitchId = $Object.Switch
            $This.Image    = $Object.Image
        }
        VmControllerTemplate()
        {
            $This.Index    = $Null
            $This.Guid     = $This.NewGuid()
            $This.Name     = "<New>"
        }
        [Object] NewGuid()
        {
            Return [Guid]::NewGuid()
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmController[Template]>"
        }
    }
```

```
PS Prompt:\> $Ctrl.Xaml.IO.TemplateOutput.Items | Format-Table

Index Guid                                 Name      Role   Base Memory Hdd      Gen Core SwitchId
----- ----                                 ----      ----   ---- ------ ---      --- ---- --------
0     0cd57e77-4251-41db-8254-7f2da1a07050 desktop01 Client             64.00 GB 2   2
      20b4e119-6f1a-49bd-af9e-0521b5ebd92a <New>

PS Prompt:\>
```

```
                                                                                     ______________________________/
\___________________________________________________________________________________/ Class [VmControllerTemplate]
  Class [VmControllerNodeSwitch] /------------------------------------------------------------------------------------\
/--------------------------------
```

Same idea as the above, strictly meant to allow [switches] to be [created] or [removed].

```
Class VmControllerNodeSwitch
{
    [String]       $Index
    [Guid]          $Guid
    [String]        $Name
    [String]        $Type
    [String] $Description
    VmControllerNodeSwitch([Object]$Object)
    {
        $This.Index       = $Object.Index
        $This.Guid        = $Object.Guid
        $This.Name        = $Object.Name
        $This.Type        = $Object.Type
        $This.Description = $Object.Description
    }
    VmControllerNodeSwitch()
    {
        $This.Index       = $Null
        $This.Guid        = $This.NewGuid()
        $This.Name        = "<New>"
    }
    [Object] NewGuid()
    {
        Return [Guid]::NewGuid()
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmController[NodeSwitch]>"
    }
}
```

```
                                                                                         ________________________________/
\______________________________________________________________________________________/ Class [VmControllerNodeSwitch]
  Class [VmControllerMaster] /-----------------------------------------------------------------------------------------\
/----------------------------
```

```
When you need a [system] at the very [center] of it all...
...one that could manage [configurations], [virtual machines], or [reflyable buildings like the SpaceX Starship]...

...you need something that is [extremely high-fidelity].

Something that had a lot of [time], [thought], [attention], and [focus] put into it...
...so that it was [engineered to work].

That's what this class is, below.
(...some text/line wrapping...)
```

```
Class VmControllerMaster
{
    [Object]     $Module
    [Object]       $Xaml
    [Object]     $Master
    [Object] $Credential
    [Object]      $Image
    [Object]   $Template
    [Object]       $Node
    [Object]       $Flag
    VmControllerMaster()
    {
        $This.Module      = $This.GetFEModule()
        $This.Xaml        = $This.VmXaml()
        $This.Master      = $This.VmMaster()
        $This.Credential  = $This.VmCredential()
        $This.Image       = $This.ImageController()
        $This.Template    = $This.VmTemplate()
        $This.Node        = $This.VmNode()
        $This.Flag        = @( )
        ForEach ($Name in "MasterPath",
                        "MasterDomain",
                        "MasterNetBios",
                        "CredentialUsername",
                        "CredentialPassword",
                        "CredentialConfirm",
                        "CredentialPin",
                        "ImagePath",
                        "TemplateName",
                        "TemplatePath",
                        "TemplateImagePath",
                        "NodeTemplatePath")
        {
            $This.Flag += $This.VmControllerFlag($This.Flag.Count,$Name)
        }
    }
    Update([Int32]$State,[String]$Status)
    {
        # Updates the console
        $This.Module.Update($State,$Status)
    }
    Error([UInt32]$State,[String]$Status)
    {
        $This.Module.Update($State,$Status)
        Throw $This.Module.Console.Last().Status
    }
    DumpConsole()
    {
        $xPath = "{0}\{1}-{2}.log" -f $This.LogPath(), $This.Now(), $This.Name
        $This.Update(100,"[+] Dumping console: [$xPath]")
        $This.Console.Finalize()

        $Value = $This.Console.Output | % ToString

        [System.IO.File]::WriteAllLines($xPath,$Value)
    }
    [String] LogPath()
    {
        $xPath = $This.ProgramData()

        ForEach ($Folder in $This.Author(), "Logs")
        {
            $xPath = $xPath, $Folder -join "\"
            If (![System.IO.Directory]::Exists($xPath))
            {
                [System.IO.Directory]::CreateDirectory($xPath)
            }
        }

        Return $xPath
    }
    [String] Now()
    {
        Return [DateTime]::Now.ToString("yyyy-MMdd_HHmmss")
    }
    [String] ProgramData()
    {
        Return [Environment]::GetEnvironmentVariable("ProgramData")
    }
    [String] Author()
    {
        Return "Secure Digits Plus LLC"
    }
    [Object] GetFEModule()
    {
        $Item = Get-FEModule -Mode 1
        $Item.Console.Reset()
        $Item.Mode = 0
        $Item.Console.Initialize()
        Return $Item
    }
```

```
        [Object] VmXaml()
        {
            $This.Update(0,"Getting [~] VmXaml")
            Return [XamlWindow][VmControllerXaml]::Content
        }
        [Object] VmMaster()
        {
            $This.Update(0,"Getting [~] VmMaster")
            Return [VmNetworkMaster]::New()
        }
        [Object] VmCredential()
        {
            $This.Update(0,"Getting [~] VmCredential")
            Return [VmCredentialMaster]::New()
        }
        [Object] VmTemplate()
        {
            $This.Update(0,"Getting [~] VmTemplate")
            Return [VmTemplateMaster]::New()
        }
        [Object] VmNode()
        {
            $This.Update(0,"Getting [~] VmNode")
            Return [VmNodeMaster]::New()
        }
        [Object] ImageController()
        {
            $This.Update(0,"Getting [~] ImageController")
            Return [ImageController]::New()
        }
        [Object] VmControllerFlag([UInt32]$Index,[String]$Name)
        {
            Return [VmControllerFlag]::New($Index,$Name)
        }
        [Object] VmControllerProperty([Object]$Property)
        {
            Return [VmControllerProperty]::New($Property)
        }
        [Object] Grid([String]$Name)
        {
            $Item = Switch ($Name)
            {
                VmControllerCredential   {   [VmControllerCredential]::New() }
                VmControllerTemplate     {     [VmControllerTemplate]::New() }
                VmControllerNodeSwitch   {   [VmControllerNodeSwitch]::New() }
            }

            Return $Item
        }
        [Object] Grid([String]$Name,[Object]$Object)
        {
            $Item = Switch ($Name)
            {
                VmControllerCredential   {   [VmControllerCredential]::New($Object) }
                VmControllerTemplate     {     [VmControllerTemplate]::New($Object) }
                VmControllerNodeSwitch   {   [VmControllerNodeSwitch]::New($Object) }
            }

            Return $Item
        }
        [Object[]] Control([UInt32]$Index)
        {
            $Out  = @( )
            $Slot = Switch ($Index)
            {
                0 { $This.Credential.Output }
                1 { $This.Template.Output   }
                2 { $This.Node.Switch       }
            }

            $Id   = Switch ($Index)
            {
                0 { "VmControllerCredential"   }
                1 { "VmControllerTemplate"     }
                2 { "VmControllerNodeSwitch"   }
            }

            ForEach ($Item in $Slot)
            {
                $Out += $This.Grid($Id,$Item)
            }

            $Out += $This.Grid($Id)

            Return $Out
        }
        SetNetwork([UInt32]$Index)
        {
            $This.Update(0,"Setting [~] Network")
            $This.Master.SetNetwork($Index)

            $This.PingSweep($This.Master.Network.Hosts)

            $This.Update(0,"Setting [~] Dhcp")
            $This.Master.Network.SetDhcp()
        }
        SetImagePath([String]$Path)
        {
            $This.Update(0,"Setting [~] Image source")
            $This.Image.SetSource($Path)
            $This.Image.Refresh()
            $This.Reset($This.Xaml.IO.ImageStore,$This.Image.Store)

            Switch ($This.Image.Store.Count)
            {
                0
                {
                    Throw "No images detected"
                }
                1
                {
                    $This.Image.Select(0)
                    $This.Update(0,"Processing [~] $($This.Image.Current().Name)")
                    $This.Image.ProcessSlot()
                }
                Default
                {
                    ForEach ($X in 0..($This.Image.Store.Count-1))
                    {
                        $This.Image.Select($X)
                        $This.Update(0,"Processing [~] $($This.Image.Current().Name)")
                        $This.Image.ProcessSlot()
                    }
                }
            }

            $This.Update(1,"Complete [+] Images charted")
        }
        PingSweep([Object[]]$Range)
        {
            $This.Update(0,"Scanning [~] Network host(s)")
            $Hosts        = $Range.IpAddress
            $RS           = [System.Management.Automation.Runspaces.RunspaceFactory]::CreateRunspace()
            $PS           = [PowerShell]::Create()
            $PS.Runspace  = $RS
            $RS.Open()
            [Void]$PS.AddScript({

                Param ($Hosts)

                $Buffer   = 97..119 + 97..105 | % { "0x{0:X}" -f $_ }
                $Option   = New-Object System.Net.NetworkInformation.PingOptions
                $Ping     = @{ }
                ForEach ($X in 0..($Hosts.Count-1))
                {
                    $Item = New-Object System.Net.NetworkInformation.Ping
                    $Ping.Add($X,$Item.SendPingAsync($Hosts[$X],100,$Buffer,$Option))
                }

                $Ping[0..($Ping.Count-1)]
            })

            $PS.AddArgument($Hosts)
            $Async        = $PS.BeginInvoke()
            $Output       = $PS.EndInvoke($Async)
            $PS.Dispose()
            $RS.Dispose()

            $This.Update(0,"Scanned [+] Network host(s), resolving hostnames")
            ForEach ($X in 0..($Output.Count-1))
            {
                $Status           = [UInt32]($Output[$X].Result.Status -eq "Success")
                $Range[$X].Status = $Status
                If ($Status -eq 1)
                {
                    $Range[$X].Resolve()
                }
            }
        }
        FolderBrowse([String]$Name)
        {
            $This.Update(0,"Browsing [~] Folder: [$Name]")
            $Object      = $This.Xaml.Get($Name)
            $Item        = New-Object System.Windows.Forms.FolderBrowserDialog
            $Item.ShowDialog()

            $Object.Text = @("<Select a path>",$Item.SelectedPath)[!!$Item.SelectedPath]
        }
        FileBrowse([String]$Name)
        {
            $This.Update(0,"Browsing [~] File: [$Name]")
            $Object      = $This.Xaml.Get($Name)
            $Item                   = New-Object System.Windows.Forms.OpenFileDialog
            $Item.InitialDirectory  = $Env:SystemDrive
            $Item.ShowDialog()

            If (!$Item.Filename)
            {
                $Item.Filename                = ""
            }

            $Object.Text = @("<Select an image>",$Item.FileName)[!!$Item.FileName]
        }
        [String[]] Reserved()
        {
            Return "ANONYMOUS;AUTHENTICATED USER;BATCH;BUILTIN;CREATOR GROUP;CREATOR GR"+
            "OUP SERVER;CREATOR OWNER;CREATOR OWNER SERVER;DIALUP;DIGEST AUTH;IN"+
            "TERACTIVE;INTERNET;LOCAL;LOCAL SYSTEM;NETWORK;NETWORK SERVICE;NT AU"+
            "THORITY;NT DOMAIN;NTLM AUTH;NULL;PROXY;REMOTE INTERACTIVE;RESTRICTE"+
            "D;SCHANNEL AUTH;SELF;SERVER;SERVICE;SYSTEM;TERMINAL SERVER;THIS ORG"+
            "ANIZATION;USERS;WORLD" -Split ";"
        }
        [String[]] Legacy()
        {
            Return "-GATEWAY;-GW;-TAC" -Split ";"
        }
        [String[]] SecurityDescriptor()
        {
            Return "AN;AO;AU;BA;BG;BO;BU;CA;CD;CG;CO;DA;DC;DD;DG;DU;EA;ED;HI;IU;"+
            "LA;LG;LS;LW;ME;MU;NO;NS;NU;PA;PO;PS;PU;RC;RD;RE;RO;RS;RU;SA;SI;SO;S"+
            "U;SY;WD" -Split ";"
        }
        [String] IconStatus([UInt32]$Flag)
        {
            Return $This.Module._Control(@("failure.png","success.png")[$Flag]).Fullname
        }
```

```
        ToggleMasterCreate()
        {
            $C = 0
            $D = 0
            ForEach ($Item in $This.Flag | ? Name -match "^Master")
            {
                If ($Item.Status -eq 1)
                {
                    $C ++
                }
            }

            If ($This.Xaml.IO.MasterConfig.SelectedIndex -ne -1)
            {
                $D = 1
            }

            $This.Xaml.IO.MasterCreate.IsEnabled = $C -eq 3 -and $D -eq 1
        }
        CheckUsername()
        {
            $Username      = $This.Xaml.IO.CredentialUsername.Text
            $xFlag         = $This.Flag | ? Name -eq CredentialUsername
            $xFlag.Status = [UInt32]($Username -ne "" -and $Username -notin $This.Credential.Output)

            $This.Xaml.IO.CredentialUsernameIcon.Source = $This.IconStatus($xFlag.Status)
        }
        CheckPassword()
        {
            $Password      = $This.Xaml.IO.CredentialPassword.Password
            $xFlag         = $This.Flag | ? Name -eq CredentialPassword
            $xFlag.Status = [UInt32]($Password -ne "")

            $This.Xaml.IO.CredentialPasswordIcon.Source = $This.IconStatus($xFlag.Status)
        }
        CheckConfirm()
        {
            $Password      = [Regex]::Escape($This.Xaml.IO.CredentialPassword.Password)
            $Confirm       = [Regex]::Escape($This.Xaml.IO.CredentialConfirm.Password)
            $xFlag         = $This.Flag | ? Name -eq CredentialConfirm
            $xFlag.Status = [UInt32]($Password -ne "" -and $Password -eq $Confirm)

            $This.Xaml.IO.CredentialConfirmIcon.Source  = $This.IconStatus($xFlag.Status)
        }
        CheckPin()
        {
            $Pin           = $This.Xaml.IO.CredentialPin.Password
            $xFlag         = $This.Flag | ? Name -eq CredentialPin
            $xFlag.Status = [UInt32]($Pin.Length -ge 4)

            $This.Xaml.IO.CredentialPinIcon.Source      = $This.IconStatus($xFlag.Status)
        }
        ToggleCredentialCreate()
        {
            $Mode = [UInt32]($This.Xaml.IO.CredentialType.SelectedIndex -eq 4)

            Switch ($Mode)
            {
                0
                {
                    $This.CheckUsername()
                    $This.CheckPassword()
                    $This.CheckConfirm()

                    $C = 0
                    ForEach ($Item in $This.Flag | ? Name -match "^Credential")
                    {
                        If ($Item.Status -eq 1)
                        {
                            $C ++
                        }
                    }
                    $This.Xaml.IO.CredentialCreate.IsEnabled = [UInt32]($C -eq 3)
                }
                1
                {
                    $This.CheckUsername()
                    $This.CheckPassword()
                    $This.CheckConfirm()
                    $This.CheckPin()

                    $C = 0
                    ForEach ($Item in $This.Flag | ? Name -match "^Credential")
                    {
                        If ($Item.Status -eq 1)
                        {
                            $C ++
                        }
                    }

                    $This.Xaml.IO.CredentialCreate.IsEnabled = [UInt32]($C -eq 4)
                }
            }
        }
        ToggleTemplateCreate()
        {
            $C = 0
            ForEach ($Item in $This.Flag | ? Name -match "^Template")
            {
                If ($Item.Status -eq 1)
                {
                    $C ++
                }
            }

            $This.Xaml.IO.TemplateCreate.IsEnabled = $C -eq 3
        }
        CheckPath([String]$Name)
        {
            $Item        = $This.Xaml.Get($Name)
            $Icon        = $This.Xaml.Get("$Name`Icon")

            $xFlag       = $This.Flag | ? Name -eq $Name
            $xFlag.SetStatus([UInt32][System.IO.Directory]::Exists($Item.Text))

            $Icon.Source = $This.IconStatus($xFlag.Status)

            $This.ToggleMasterCreate()
        }
        CheckDomain()
        {
            $Item = $This.Xaml.IO.MasterDomain.Text

            If ($Item.Length -lt 2 -or $Item.Length -gt 63)
            {
                $X = "[!] Length not between 2 and 63 characters"
            }
            ElseIf ($Item -in $This.Reserved())
            {
                $X = "[!] Entry is in reserved words list"
            }
            ElseIf ($Item -in $This.Legacy())
            {
                $X = "[!] Entry is in the legacy words list"
            }
            ElseIf ($Item -notmatch "(?=^.{4,253}$)(^((?!-)[a-zA-Z0-9-]{1,63}(?<!-)\.)+[a-zA-Z]{2,63}$)")
            {
                $X = "[!] Invalid characters"
            }
            ElseIf ($Item[0,-1] -match "(\W)")
            {
                $X = "[!] First/Last Character cannot be a '.' or '-'"
            }
            ElseIf ($Item.Split(".").Count -lt 2)
            {
                $X = "[!] Single label domain names are disabled"
            }
            ElseIf ($Item.Split('.')[-1] -notmatch "\w")
            {
                $X = "[!] Top Level Domain must contain a non-numeric"
            }
            Else
            {
                $X = "[+] Passed"
            }

            $xFlag = $This.Flag | ? Name -eq MasterDomain
            $xFlag.SetStatus([UInt32]($X -eq "[+] Passed"))

            $This.Xaml.IO.MasterDomainIcon.Source = $This.IconStatus($xFlag.Status)

            $This.ToggleMasterCreate()
        }
        CheckNetBios()
        {
            $Item = $This.Xaml.IO.MasterNetBios.Text

            If ($Item.Length -lt 1 -or $Item.Length -gt 15)
            {
                $X = "[!] Length not between 1 and 15 characters"
            }
            ElseIf ($Item -in $This.Reserved())
            {
                $X = "[!] Entry is in reserved words list"
            }
            ElseIf ($Item -in $This.Legacy())
            {
                $X = "[!] Entry is in the legacy words list"
            }
            ElseIf ($Item -notmatch "([\.\-0-9a-zA-Z])")
            {
                $X = "[!] Invalid characters"
            }
            ElseIf ($Item[0,-1] -match "(\W)")
            {
                $X = "[!] First/Last Character cannot be a '.' or '-'"
            }
            ElseIf ($Item -match "\.")
            {
                $X = "[!] NetBIOS cannot contain a '.'"
            }
            ElseIf ($Item -in $This.SecurityDescriptor())
            {
                $X = "[!] Matches a security descriptor"
            }
            Else
            {
                $X = "[+] Passed"
            }

            $xFlag = $This.Flag | ? Name -eq MasterNetBios
            $xFlag.SetStatus([UInt32]($X -eq "[+] Passed"))

            $This.Xaml.IO.MasterNetBiosIcon.Source = $This.IconStatus($xFlag.Status)

            $This.ToggleMasterCreate()
        }
        CheckTemplateName()
        {
            $Item          = $This.Xaml.Get("TemplateName")
            $xFlag         = $This.Flag | ? Name -eq TemplateName
            $xFlag.Status = [UInt32]($Item.Text -match "[a-zA-Z]{1}[a-zA-Z0-9]{0,14}" -and $Item.Text -notin
    $This.Node.Host.Name)

            $This.Xaml.IO.TemplateNameIcon.Source = $This.IconStatus($xFlag.Status)
            $This.ToggleTemplateCreate()
        }
        CheckTemplatePath()
        {
            $Item         = $This.Xaml.Get("TemplatePath")
            $xFlag        = $This.Flag | ? Name -eq TemplatePath
            $xFlag.Status = [UInt32][System.IO.Directory]::Exists($Item.Text)

            $This.Xaml.IO.TemplatePathIcon.Source = $This.IconStatus($xFlag.Status)

            $This.ToggleTemplateCreate()
        }
        CheckTemplateImagePath()
        {
            $Item         = $This.Xaml.Get("TemplateImagePath")
            $xFlag        = $This.Flag | ? Name -eq TemplateImagePath
            $xFlag.Status = [UInt32][System.IO.File]::Exists($Item.Text)

            $This.Xaml.IO.TemplateImagePathIcon.Source = $This.IconStatus($xFlag.Status)

            $This.ToggleTemplateCreate()
        }
        CheckNodeSwitchName()
        {
            $Item         = $This.Xaml.Get("NodeSwitchName")
            $xFlag        = $This.Flag | ? Name -eq NodeSwitchIcon
            $xFlag.Status = [UInt32][System.IO.Directory]::Exists($Item.Text)

            $This.Xaml.IO.NodeSwitchNameIcon.Source = $This.IconStatus($xFlag.Status)
        }
        CheckNodeTemplatePath()
        {
            $Item         = $This.Xaml.Get("NodeTemplatePath")
            $xFlag        = $This.Flag | ? Name -eq "NodeTemplatePath"
            $xFlag.Status = [UInt32][System.IO.Directory]::Exists($Item.Text)

            $This.Xaml.IO.NodeTemplatePathIcon.Source = $This.IconStatus($xFlag.Status)
        }
        Reset([Object]$xSender,[Object]$Object)
        {
            $xSender.Items.Clear()
            ForEach ($Item in $Object)
            {
                $xSender.Items.Add($Item)
            }
        }
        [Object[]] Property([Object]$Object)
        {
            Return $Object.PSObject.Properties | % { $This.VmControllerProperty($_) }
        }
        [Object[]] Property([Object]$Object,[UInt32]$Mode,[String[]]$Property)
        {
            $Item = Switch ($Mode)
            {
                0 { $Object.PSObject.Properties | ? Name -notin $Property }
                1 { $Object.PSObject.Properties | ? Name    -in $Property }
            }

            Return $Item | % { $This.VmControllerProperty($_) }
        }
        SetInitialState()
        {
            # Master panel
            $This.Xaml.IO.MasterPath.Text              = "<Select a path>"
            $This.Xaml.IO.MasterCreate.IsEnabled       = 0

            # Credential panel
            $This.Xaml.IO.CredentialType.SelectedIndex = 0
            $This.Reset($This.Xaml.IO.CredentialDescription,$This.Credential.Slot[0])

            $This.Xaml.IO.CredentialRemove.IsEnabled   = 0
            $This.Xaml.IO.CredentialCreate.IsEnabled   = 0

            # Image panel
            $This.Xaml.IO.ImageImport.IsEnabled        = 0

            # Template panel
            $This.Xaml.IO.TemplateCreate.IsEnabled     = 0
            $This.Xaml.IO.TemplateRemove.IsEnabled     = 0
            $This.Xaml.IO.TemplateExport.IsEnabled     = 0
            $This.Xaml.IO.TemplateCredentialCount.Text = $This.Credential.Output.Count

            $This.Xaml.IO.TemplateRole.SelectedIndex   = 0
            $This.Xaml.IO.TemplateSwitch.SelectedIndex = 0

            $This.Xaml.IO.TemplateOutput.SelectedIndex = $This.Template.Output.Count

            # Node panel
            $This.Xaml.IO.NodeSwitchCreate.IsEnabled   = 0
            $This.Xaml.IO.NodeSwitchRemove.IsEnabled   = 0

            $This.Xaml.IO.NodeHostCreate.IsEnabled     = 0
            $This.Xaml.IO.NodeHostRemove.IsEnabled     = 0

            $This.Xaml.IO.NodeSlot.SelectedIndex       = 1
            $This.Xaml.IO.NodeTemplateImport.IsEnabled = 0

            $This.Update(0,"Complete [+] Initial GUI state")
        }
        CredentialPanel()
        {
            $This.Xaml.IO.CredentialCreate.IsEnabled       = 0
            $This.Xaml.IO.CredentialRemove.IsEnabled       = 0
            $This.Xaml.IO.CredentialType.IsEnabled         = 0
            $This.Xaml.IO.CredentialDescription.IsEnabled  = 0
            $This.Xaml.IO.CredentialUsername.IsEnabled     = 0
            $This.Xaml.IO.CredentialPassword.IsEnabled     = 0
            $This.Xaml.IO.CredentialConfirm.IsEnabled      = 0
            $This.Xaml.IO.CredentialPin.IsEnabled          = $This.Xaml.IO.CredentialType.SelectedIndex -eq 4

            $This.Xaml.IO.CredentialUsername.Text          = ""
            $This.Xaml.IO.CredentialPassword.Password      = ""
            $This.Xaml.IO.CredentialConfirm.Password       = ""
            $This.Xaml.IO.CredentialPin.Password           = ""

            $This.Xaml.IO.CredentialUsernameIcon.Source    = $Null
            $This.Xaml.IO.CredentialPasswordIcon.Source    = $Null
            $This.Xaml.IO.CredentialConfirmIcon.Source     = $Null
            $This.Xaml.IO.CredentialPinIcon.Source         = $Null

            If ($This.Xaml.IO.CredentialOutput.SelectedIndex -ne -1)
            {
                $This.Xaml.IO.CredentialUsername.IsEnabled = 1
                $This.Xaml.IO.CredentialPassword.IsEnabled = 1
                $This.Xaml.IO.CredentialConfirm.IsEnabled  = 1

                $Selected = $This.Xaml.IO.CredentialOutput.SelectedItem
                $Item     = $This.Credential.Output | ? Guid -eq $Selected.Guid
                If (!!$Item)
                {
                    $This.Xaml.IO.CredentialType.SelectedIndex    = $This.Credential.Slot | ? Name -eq
    $Selected.Type | % Index
                    $This.Xaml.IO.CredentialUsername.Text         = $Item.Username
                    $This.Xaml.IO.CredentialPassword.Password     = $Item.Password()
                    $This.Xaml.IO.CredentialConfirm.Password      = $Item.Password()
                    $This.Xaml.IO.CredentialCreate.IsEnabled      = 0
                    $This.Xaml.IO.CredentialRemove.IsEnabled      = 1
                }
                Else
                {
                    $This.Xaml.IO.CredentialUsername.Text         = ""
                    $This.Xaml.IO.CredentialPassword.Password     = ""
                    $This.Xaml.IO.CredentialConfirm.Password      = ""
                    $This.Xaml.IO.CredentialType.IsEnabled        = 1
                    $This.Xaml.IO.CredentialDescription.IsEnabled = 1
                }

                If ($Item.Type -eq "Microsoft")
                {
                    $This.Xaml.IO.CredentialPin.Password          = $Item.Pin
                }
            }
        }
        TemplatePanel()
        {
            $This.Xaml.IO.TemplateCreate.IsEnabled              = 0
            $This.Xaml.IO.TemplateRemove.IsEnabled              = 0
            $This.Xaml.IO.TemplateExport.IsEnabled              = 0
            $This.Xaml.IO.TemplateName.IsEnabled                = 0
            $This.Xaml.IO.TemplateRole.IsEnabled                = 0
            $This.Xaml.IO.TemplatePath.IsEnabled                = 0
            $This.Xaml.IO.TemplatePathIcon.IsEnabled            = 0
            $This.Xaml.IO.TemplatePathBrowse.IsEnabled          = 0
            $This.Xaml.IO.TemplateMemory.IsEnabled              = 0
            $This.Xaml.IO.TemplateHardDrive.IsEnabled           = 0
            $This.Xaml.IO.TemplateGeneration.IsEnabled          = 0
            $This.Xaml.IO.TemplateCore.IsEnabled                = 0
            $This.Xaml.IO.TemplateSwitch.IsEnabled              = 0
            $This.Xaml.IO.TemplateImagePath.IsEnabled           = 0
            $This.Xaml.IO.TemplateImagePathIcon.IsEnabled       = 0
            $This.Xaml.IO.TemplateImagePathBrowse.IsEnabled     = 0

            $This.Xaml.IO.TemplateMemory.SelectedIndex          = 1
            $This.Xaml.IO.TemplateHardDrive.SelectedIndex       = 1
            $This.Xaml.IO.TemplateGeneration.SelectedIndex      = 1
            $This.Xaml.IO.TemplateCore.SelectedIndex            = 1

            $This.Xaml.IO.TemplatePathIcon.Source               = $Null
            $This.Xaml.IO.TemplateImagePathIcon.Source          = $Null

            If ($This.Xaml.IO.TemplateOutput.SelectedIndex -ne -1)
            {
                $This.Xaml.IO.TemplateName.IsEnabled            = 1
                $This.Xaml.IO.TemplateRole.IsEnabled            = 1
                $This.Xaml.IO.TemplatePath.IsEnabled            = 1
                $This.Xaml.IO.TemplatePathIcon.IsEnabled        = 1
                $This.Xaml.IO.TemplatePathBrowse.IsEnabled      = 1
                $This.Xaml.IO.TemplateMemory.IsEnabled          = 1
                $This.Xaml.IO.TemplateHardDrive.IsEnabled       = 1
                $This.Xaml.IO.TemplateGeneration.IsEnabled      = 1
                $This.Xaml.IO.TemplateCore.IsEnabled            = 1
                $This.Xaml.IO.TemplateSwitch.IsEnabled          = 1
                $This.Xaml.IO.TemplateImagePath.IsEnabled       = 1
                $This.Xaml.IO.TemplateImagePathIcon.IsEnabled   = 1
                $This.Xaml.IO.TemplateImagePathBrowse.IsEnabled = 1

                $Selected = $This.Xaml.IO.TemplateOutput.SelectedItem
                $Item     = $This.Template.Output | ? Guid -eq $Selected.Guid
                If (!!$Item)
                {
                    $This.Xaml.IO.TemplateCreate.IsEnabled          = 0
                    $This.Xaml.IO.TemplateRemove.IsEnabled          = 1
                    $This.Xaml.IO.TemplateExport.IsEnabled          = 1
                    $This.Xaml.IO.TemplateName.Text                 = $Item.Name
                    $This.Xaml.IO.TemplateRole.SelectedIndex        = $Item.Role.Index
                    $This.Xaml.IO.TemplatePath.Text                 = $Item.Base
                    $This.Xaml.IO.TemplateMemory.SelectedIndex      = Switch ($Item.Memory)
                    {
                        "2.00 GB"   { 0 }
                        "4.00 GB"   { 1 }
                        "8.00 GB"   { 2 }
                        "16.00 GB"  { 3 }
                    }
                    $This.Xaml.IO.TemplateHardDrive.SelectedIndex   = Switch ($Item.Hdd)
                    {
                        "32.00 GB"  { 0 }
                        "64.00 GB"  { 1 }
                        "128.00 GB" { 2 }
                        "256.00 GB" { 3 }
                    }
                    $This.Xaml.IO.TemplateGeneration.SelectedIndex  = @{"1"=0;"2"=1}[$Item.Gen]
                    $This.Xaml.IO.TemplateCore.SelectedIndex        = @{"1"=0;"2"=1;"3"=2;"4"=3}[$Item.Core]
                    $This.Xaml.IO.TemplateSwitch.SelectedIndex      = $This.Node.Switch | ? Name -eq 
    $Item.SwitchId | % Index
                    $This.Xaml.IO.TemplateImagePath.Text            = $Item.Image.File.Fullname
                    $This.Xaml.IO.TemplateCreate.IsEnabled          = 0
                }
                Else
                {
                    $This.Xaml.IO.TemplateName.Text                 = ""
                    $This.Xaml.IO.TemplateRole.SelectedIndex        = 1
                    $This.Xaml.IO.TemplatePath.Text                 = "<Select a path>"
                    $This.Xaml.IO.TemplateImagePath.Text            = "<Select an image>"
                }
            }
        }
        NodeSwitchPanel()
        {
            $This.Xaml.IO.NodeSwitchCreate.IsEnabled = 0
            $This.Xaml.IO.NodeSwitchRemove.IsEnabled = 0
            $This.Xaml.IO.NodeSwitchUpdate.IsEnabled = 1

            $This.Xaml.IO.NodeSwitchIcon.Source      = $Null

            If ($This.Xaml.IO.TemplateOutput.SelectedIndex -ne -1)
            {
                $Selected = $This.Xaml.IO.NodeSwitch.SelectedItem
                $Item     = $This.Node.Switch | ? Guid -eq $Selected.Guid
                If (!!$Item)
                {
                    $This.Xaml.IO.NodeSwitchRemove.IsEnabled = 1
                }
            }
        }
        NodeHostPanel()
        {
            $This.Xaml.IO.NodeHostCreate.IsEnabled = 0
            $This.Xaml.IO.NodeHostRemove.IsEnabled = 0
            $This.Xaml.IO.NodeHostUpdate.IsEnabled = 1

            If ($This.Xaml.IO.NodeHost.SelectedIndex -ne -1)
            {
                $Selected = $This.Xaml.IO.NodeHost.SelectedItem
                $Mode     = $Selected.Type -eq "Template"
                $Slot     = @($This.Node.Host,$This.Node.Template)[$Mode]
                $Item     = $Slot | ? Guid -eq $Selected.Guid
                $This.Reset($This.Xaml.IO.NodeHostExtension,$This.Property($Item))

                $This.Xaml.IO.NodeHostCreate.IsEnabled = $Mode
                $This.Xaml.IO.NodeHostRemove.IsEnabled = 1
            }
        }
        Invoke()
        {
            Try
            {
                $This.Xaml.Invoke()
            }
            Catch
            {
                $This.Module.Write(1,"Failed [!] Either the user cancelled or the dialog failed")
            }
        }
        StageXaml()
        {
            $Ctrl = $This
          <#
              ____    __________________________________________________________________________________________
             //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
             \\__//¯¯¯ Master [~] Panel                                                                 ___//¯¯\\
              ¯¯¯\\__________________________________________________________________________________________//¯¯\\__//
                  ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ____
          #>

            $Ctrl.Reset($Ctrl.Xaml.IO.MasterConfig,$Ctrl.Master.Config)
            $Ctrl.Xaml.IO.MasterConfig.Add_SelectionChanged(
            {
                $Ctrl.ToggleMasterCreate()
            })

            $Ctrl.Xaml.IO.MasterPath.Add_TextChanged(
            {
                $Ctrl.CheckPath("MasterPath")
            })

            $Ctrl.Xaml.IO.MasterPathBrowse.Add_Click(
            {
                $Ctrl.FolderBrowse("MasterPath")
            })

            $Ctrl.Xaml.IO.MasterDomain.Add_TextChanged(
            {
                $Ctrl.CheckDomain()
            })

            $Ctrl.Xaml.IO.MasterNetBios.Add_TextChanged(
            {
                $Ctrl.CheckNetBios()
            })

            $Ctrl.Xaml.IO.MasterCreate.Add_Click(
            {
                $Ctrl.Master.SetMain($Ctrl.Xaml.IO.MasterPath.Text,
                                    $Ctrl.Xaml.IO.MasterDomain.Text,
                                    $Ctrl.Xaml.IO.MasterNetBios.Text)

                $Ctrl.SetNetwork($Ctrl.Xaml.IO.MasterConfig.SelectedIndex)

                ForEach ($Item in "Config","Path","Domain","NetBios","PathBrowse","Create")
                {
                    $Ctrl.Xaml.Get("Master$Item").IsEnabled = 0
                }

                $Ctrl.Reset($Ctrl.Xaml.IO.MasterConfigOutput,$Ctrl.Property($Ctrl.Master.Network.Config))
                $Ctrl.Reset($Ctrl.Xaml.IO.MasterBase,$Ctrl.Property($Ctrl.Master.Network.Base))
                $Ctrl.Reset($Ctrl.Xaml.IO.MasterRange,$Ctrl.Master.Network.Range)
                $Ctrl.Reset($Ctrl.Xaml.IO.MasterHosts,$Ctrl.Master.Network.Hosts)
                $Ctrl.Reset($Ctrl.Xaml.IO.MasterDhcp,$Ctrl.Property($Ctrl.Master.Network.Dhcp))
            })

          <#
                ____    __________________________________________________________________________________________
             //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
             \\__//¯¯¯ Credential [~] Panel                                                             ___//¯¯\\
                ¯¯¯\\__________________________________________________________________________________________//¯¯\\__//
                    ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ____
          #>

            $Ctrl.Xaml.IO.CredentialType.Add_SelectionChanged(
            {
                $Ctrl.Reset($Ctrl.Xaml.IO.CredentialDescription,
    $Ctrl.Credential.Slot[$Ctrl.Xaml.IO.CredentialType.SelectedIndex])
                $Ctrl.CredentialPanel()
            })

            $Ctrl.Xaml.IO.CredentialUsername.Add_TextChanged(
            {
                $Ctrl.ToggleCredentialCreate()
            })

            $Ctrl.Xaml.IO.CredentialPassword.Add_PasswordChanged(
            {
                $Ctrl.ToggleCredentialCreate()
            })

            $Ctrl.Xaml.IO.CredentialConfirm.Add_PasswordChanged(
            {
                $Ctrl.ToggleCredentialCreate()
            })

            $Ctrl.Xaml.IO.CredentialPin.Add_PasswordChanged(
            {
                $Ctrl.ToggleCredentialCreate()
            })

            $Ctrl.Xaml.IO.CredentialGenerate.Add_Click(
            {
                $Entry                                    = $Ctrl.Credential.Generate()
                $Ctrl.Xaml.IO.CredentialPassword.Password = $Entry
                $Ctrl.Xaml.IO.CredentialConfirm.Password  = $Entry
            })

            $Ctrl.Xaml.IO.CredentialOutput.Add_SelectionChanged(
            {
                $Ctrl.CredentialPanel()
            })

            $Ctrl.Xaml.IO.CredentialRemove.Add_Click(
            {
                Switch ($Ctrl.Credential.Output.Count)
                {
                    {$_ -eq 0}
                    {
                        $Ctrl.Credential.Setup()
                    }
                    {$_ -eq 1}
                    {
                        Return [System.Windows.MessageBox]::Show("Must have at least (1) account")
                    }
                    {$_ -gt 1}
                    {
                        $Ctrl.Credential.Output = @($Ctrl.Credential.Output | ? Index -ne
    $Ctrl.Xaml.IO.CredentialOutput.SelectedIndex)
                        $Ctrl.Credential.Rerank()
                    }
                }

                $Ctrl.Reset($Ctrl.Xaml.IO.CredentialOutput,$Ctrl.Control(0))
                $Ctrl.Xaml.IO.TemplateCredentialCount.Text = $Ctrl.Credential.Output.Count
            })

            $Ctrl.Xaml.IO.CredentialCreate.Add_Click(
            {
                $Ctrl.Credential.Add($Ctrl.Xaml.IO.CredentialType.SelectedIndex,
                                    $Ctrl.Xaml.IO.CredentialUsername.Text,
                                    $Ctrl.Xaml.IO.CredentialPassword.Password)

                If ($Ctrl.Xaml.IO.CredentialType.SelectedIndex -eq 4)
                {
                    $Cred     = $Ctrl.Credential.Output | ? Username -eq
    $Ctrl.Xaml.IO.CredentialUsername.Text
                    $Cred.Pin = $Ctrl.Xaml.IO.CredentialPin.Password
                }

                $Ctrl.Credential.Rerank()
                $Ctrl.Reset($Ctrl.Xaml.IO.CredentialOutput,$Ctrl.Control(0))

                $Ctrl.Xaml.IO.TemplateCredentialCount.Text = $Ctrl.Credential.Output.Count
            })

            $Ctrl.Reset($Ctrl.Xaml.IO.CredentialOutput,$Ctrl.Control(0))

          <#
```

```
    ____    ____________________________________________________________________________________________________
   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
   \\__//¯¯¯ Image [~] Panel                                                                                 ___//¯¯\\
    ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
#>

  $Ctrl.Xaml.IO.ImagePathBrowse.Add_Click(
  {
      $Ctrl.FolderBrowse("ImagePath")
  })

  $Ctrl.Xaml.IO.ImagePath.Add_TextChanged(
  {
      $Ctrl.CheckPath("ImagePath")
      $Ctrl.Xaml.IO.ImageImport.IsEnabled = $Ctrl.Flag | ? Name -eq ImagePath | % Status
  })

  $Ctrl.Xaml.IO.ImageImport.Add_Click(
  {
      $Ctrl.SetImagePath($Ctrl.Xaml.IO.ImagePath.Text)
      $Ctrl.Reset($Ctrl.Xaml.IO.ImageStore,$Ctrl.Image.Store)
  })

  $Ctrl.Xaml.IO.ImageStore.Add_SelectionChanged(
  {
      $Ctrl.Image.Select($Ctrl.Xaml.IO.ImageStore.SelectedIndex)
      $Ctrl.Reset($Ctrl.Xaml.IO.ImageStoreContent,$Ctrl.Image.Current().Content)
      $Ctrl.Xaml.IO.TemplateImagePath.Text = $Ctrl.Image.Current().Fullname
  })

<#
    ____    ____________________________________________________________________________________________________
   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
   \\__//¯¯¯ Template [~] Panel                                                                              ___//¯¯\\
    ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
#>

  $Ctrl.Xaml.IO.TemplateName.Add_TextChanged(
  {
      $Ctrl.CheckTemplateName()
  })

  $Ctrl.Xaml.IO.TemplatePath.Add_TextChanged(
  {
      $Ctrl.CheckTemplatePath()
  })

  $Ctrl.Xaml.IO.TemplatePathBrowse.Add_Click(
  {
      $Ctrl.FolderBrowse("TemplatePath")
  })

  $Ctrl.Xaml.IO.TemplateImagePath.Add_TextChanged(
  {
      $Ctrl.CheckTemplateImagePath()
  })

  $Ctrl.Xaml.IO.TemplateImagePathBrowse.Add_Click(
  {
      $Ctrl.FileBrowse("TemplateImagePath")
  })

  $Ctrl.Xaml.IO.TemplateCreate.Add_Click(
  {
      If ($Ctrl.Xaml.IO.TemplateName.Text -notmatch "(\w|\d)")
      {
          Return [System.Windows.MessageBox]::Show("Must enter a name","Error")
      }

      ElseIf ($Ctrl.Xaml.IO.TemplateName.Text -in $Ctrl.Template.Name)
      {
          Return [System.Windows.MessageBox]::Show("Duplicate name","Error")
      }
```

```powershell
            Else
            {
                $ImageFile   = $Ctrl.Image.Store | ? Fullname -eq $Ctrl.Xaml.IO.TemplateImagePath.Text
                If ($ImageFile.Type -eq "Windows")
                {
                    $ImageObject = $Ctrl.Image.ImageObject($ImageFile,
$Ctrl.Xaml.IO.ImageStoreContent.SelectedItem)
                }
                Else
                {
                    $ImageObject = $Ctrl.Image.ImageObject($ImageFile)
                }

                $Ctrl.Template.Add($Ctrl.Xaml.IO.TemplateName.Text,
                                $Ctrl.Xaml.IO.TemplateRole.SelectedIndex,
                                $Ctrl.Xaml.IO.TemplatePath.Text,
                                $Ctrl.Xaml.IO.TemplateMemory.SelectedItem.Content,
                                $Ctrl.Xaml.IO.TemplateHardDrive.SelectedItem.Content,
                                $Ctrl.Xaml.IO.TemplateGeneration.SelectedItem.Content,
                                $Ctrl.Xaml.IO.TemplateCore.SelectedItem.Content,
                                $Ctrl.Xaml.IO.TemplateSwitch.SelectedItem,
                                $ImageObject)

                $Ctrl.Reset($Ctrl.Xaml.IO.TemplateOutput,$Ctrl.Control(1))

                $Ctrl.Xaml.Get("TemplateName").Text            = ""
                $Ctrl.Xaml.Get("TemplatePath").Text            = "<Select a path>"
                $Ctrl.Xaml.Get("TemplatePathIcon").Source      = $Null
                $Ctrl.Xaml.Get("TemplateImagePath").Text       = "<Select an image>"
                $Ctrl.Xaml.Get("TemplateImagePathIcon").Source = $Null
            }
        })

        $Ctrl.Xaml.IO.TemplateOutput.Add_SelectionChanged(
        {
            $Ctrl.TemplatePanel()
        })

        $Ctrl.Xaml.IO.TemplateRemove.Add_Click(
        {
            $Ctrl.Template.Output = @($Ctrl.Template.Output | ? Name -ne
$Ctrl.Xaml.IO.TemplateOutput.SelectedItem.Name)
            $Ctrl.Reset($Ctrl.Xaml.IO.TemplateOutput,$Ctrl.Control(1))
        })

        $Ctrl.Xaml.IO.TemplateExport.Add_Click(
        {
            $Ctrl.Template.Export($Ctrl.Master.Main.Path,
                                  $Ctrl.Master.Network,
                                  $Ctrl.Credential.Output,
                                  $Ctrl.Xaml.IO.TemplateOutput.SelectedIndex)
        })

        $Ctrl.Reset($Ctrl.Xaml.IO.TemplateOutput,$Ctrl.Control(1))

      <#
          ____    _____________________________________________________________________________________________________
         //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
         \\__//¯¯¯ Node [~] Panel                                                                                ___//¯¯\\
          ¯¯¯\\___________________________________________________________________________________________________//¯¯\\__//
              ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
      #>

        $Ctrl.Xaml.IO.NodeSlot.Add_SelectionChanged(
        {
            $Ctrl.Xaml.IO.NodeSwitchPanel.Visibility = @("Collapsed","Visible")
[[UInt32]$Ctrl.Xaml.IO.NodeSlot.SelectedIndex -eq 0]
            $Ctrl.Xaml.IO.NodeHostPanel.Visibility   = @("Collapsed","Visible")
[[UInt32]$Ctrl.Xaml.IO.NodeSlot.SelectedIndex -eq 1]
        })

        $Ctrl.Xaml.IO.NodeSwitch.Add_SelectionChanged(
        {
            $Ctrl.NodeSwitchPanel()
        })

        $Ctrl.Xaml.IO.NodeSwitchUpdate.Add_Click(
        {
            $Ctrl.Node.Refresh("Switch")
            $Ctrl.Reset($Ctrl.Xaml.IO.NodeSwitch,$Ctrl.Control(2))
        })

        $Ctrl.Reset($Ctrl.Xaml.IO.NodeSwitch,$Ctrl.Control(2))
        $Ctrl.Reset($Ctrl.Xaml.IO.NodeHost,$Ctrl.Node.Host)
        $Ctrl.Reset($Ctrl.Xaml.IO.TemplateSwitch,$Ctrl.Node.Switch.Name)

        $Ctrl.Xaml.IO.NodeSwitchName.Add_TextChanged(
        {
            $Status = [UInt32]($Ctrl.Xaml.IO.NodeSwitchName.Text -notin $Ctrl.Node.Switch.Name)
            $Ctrl.Xaml.IO.NodeSwitchIcon.Source       = $Ctrl.IconStatus($Status)
            $Ctrl.Xaml.IO.NodeSwitchCreate.IsEnabled = $Status
        })

        $Ctrl.Xaml.IO.NodeSwitchCreate.Add_Click(
        {
            $Ctrl.Node.NewVmSwitch($Ctrl.Xaml.IO.NodeSwitchName.Text,
$Ctrl.Xaml.IO.NodeSwitchType.SelectedItem.Content)
            $Ctrl.Node.Refresh("Switch")
            $Ctrl.Reset($Ctrl.Xaml.IO.NodeSwitch,$Ctrl.Control(2))
        })

        $Ctrl.Xaml.IO.NodeHostUpdate.Add_Click(
        {
            $Ctrl.Node.Refresh()
            $Ctrl.Reset($Ctrl.Xaml.IO.NodeHost,$Ctrl.Node.Object)
            $Ctrl.Reset($Ctrl.Xaml.IO.NodeHostExtension,$Null)
        })

        $Ctrl.Xaml.IO.NodeTemplatePath.Add_TextChanged(
        {
            $Ctrl.CheckNodeTemplatePath()
            $Ctrl.Xaml.IO.NodeTemplateImport.IsEnabled = $Ctrl.Flag | ? Name -eq NodeTemplatePath | %
Status
        })

        $Ctrl.Xaml.IO.NodeTemplatePathBrowse.Add_Click(
        {
            $Ctrl.FolderBrowse("NodeTemplatePath")
        })

        $Ctrl.Xaml.IO.NodeTemplateImport.Add_Click(
        {
            $Ctrl.Update(0,"Setting [~] Node template import path")
            $Ctrl.Node.SetPath($Ctrl.Xaml.IO.NodeTemplatePath.Text)
            $Ctrl.Node.Refresh()
            $Ctrl.Reset($Ctrl.Xaml.IO.NodeHost,$Ctrl.Node.Object)
        })

        $Ctrl.Xaml.IO.NodeHost.Add_SelectionChanged(
        {
            $Ctrl.NodeHostPanel()
        })

        $Ctrl.Xaml.IO.NodeHostCreate.Add_Click(
        {
            $Item = $Ctrl.Xaml.IO.NodeHost.SelectedItem

            Switch ($Item.Type)
            {
                Host
                {
                    [System.Windows.MessageBox]::Show("Invalid type","Error")
                }
                Template
```

```
                    {
                        [System.Windows.MessageBox]::Show("Not yet implemented","Error")
                    }
                }
            })

            $Ctrl.Xaml.IO.NodeHostRemove.Add_Click(
            {
                $Item = $Ctrl.Xaml.IO.NodeHost.SelectedItem
                Switch ($Item.Type)
                {
                    Host
                    {
                        $xNode = $Ctrl.Node.Host | ? Guid -eq $Item.Guid
                        $Vm    = $Ctrl.Node.VmNodeObject($xNode)
                        $Vm.Remove()
                    }
                    Template
                    {
                        $xNode = Get-ChildItem $Ctrl.Node.Path | ? Name -match $Item.Name
                        Remove-Item $xNode.Fullname -Verbose
                    }
                }

                $Ctrl.Node.Refresh()
                $Ctrl.Reset($Ctrl.Xaml.IO.NodeHost,$Ctrl.Node.Object)
                $Ctrl.Reset($Ctrl.Xaml.IO.NodeHostExtension,$Null)
            })

            $Ctrl.SetInitialState()
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmController[Master]"
        }
    }
```

```
PS Prompt:\> $Ctrl

Module     : <FEModule.ModuleController>
Xaml       : <FEModule.XamlWindow[VmControllerXaml]>
Master     : <FEVirtual.VmNetwork[Master]>
Credential : <FEVirtual.VmCredential[Master]
Image      : <FEModule.Image[Controller]>
Template   : <FEVirtual.VmTemplate[Master]>
Node       : <FEVirtual.VmNode[Master]>
Flag       : {<FEVirtual.VmController[Flag]>, <FEVirtual.VmController[Flag]>, <FEVirtual.VmController[Flag]>,
             <FEVirtual.VmController[Flag]>...}

PS Prompt:\>
```

```
                                                                                          ____________________________/
\________________________________________________________________________________________/ Class [VmControllerMaster]
  Output /--------------------------------------------------------------------------------------------------------------\
/--------
```

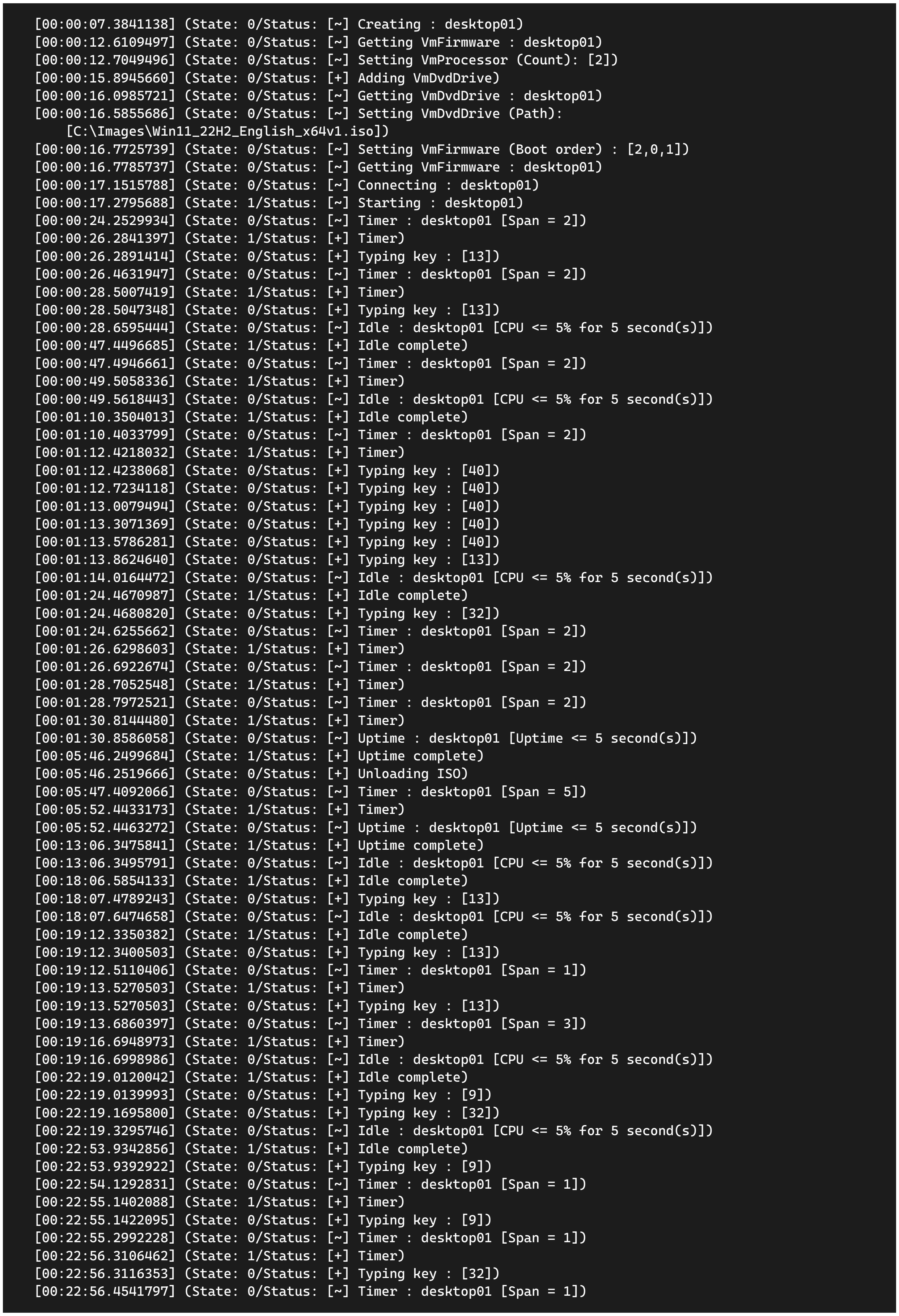
Now, here's the output that I was able to record in the video during the initial creation of this document.

This video is going to be about (6) hours long...

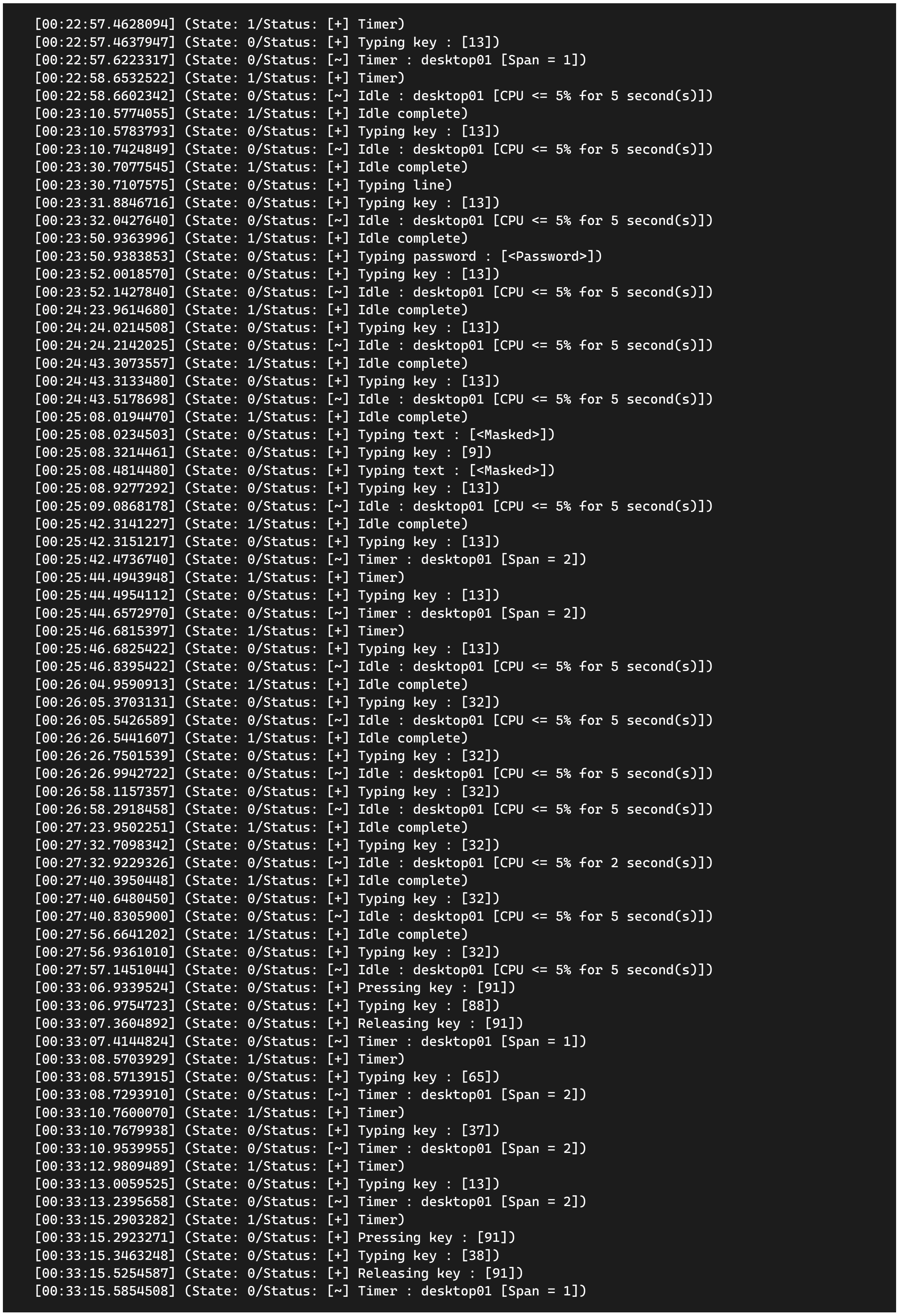
However, it will show, from [beginning] to [end], the exhition or demonstration of how this document contains the code that was [written], [uploaded], then [downloaded], and [executed] at the beginning, as well as all throughout, to show the varying objects and types that were explained up above.

There's no [Michaelsoft Deployment Toolkit] about it, really.
This thing took a lot of work, and [it's far from done].

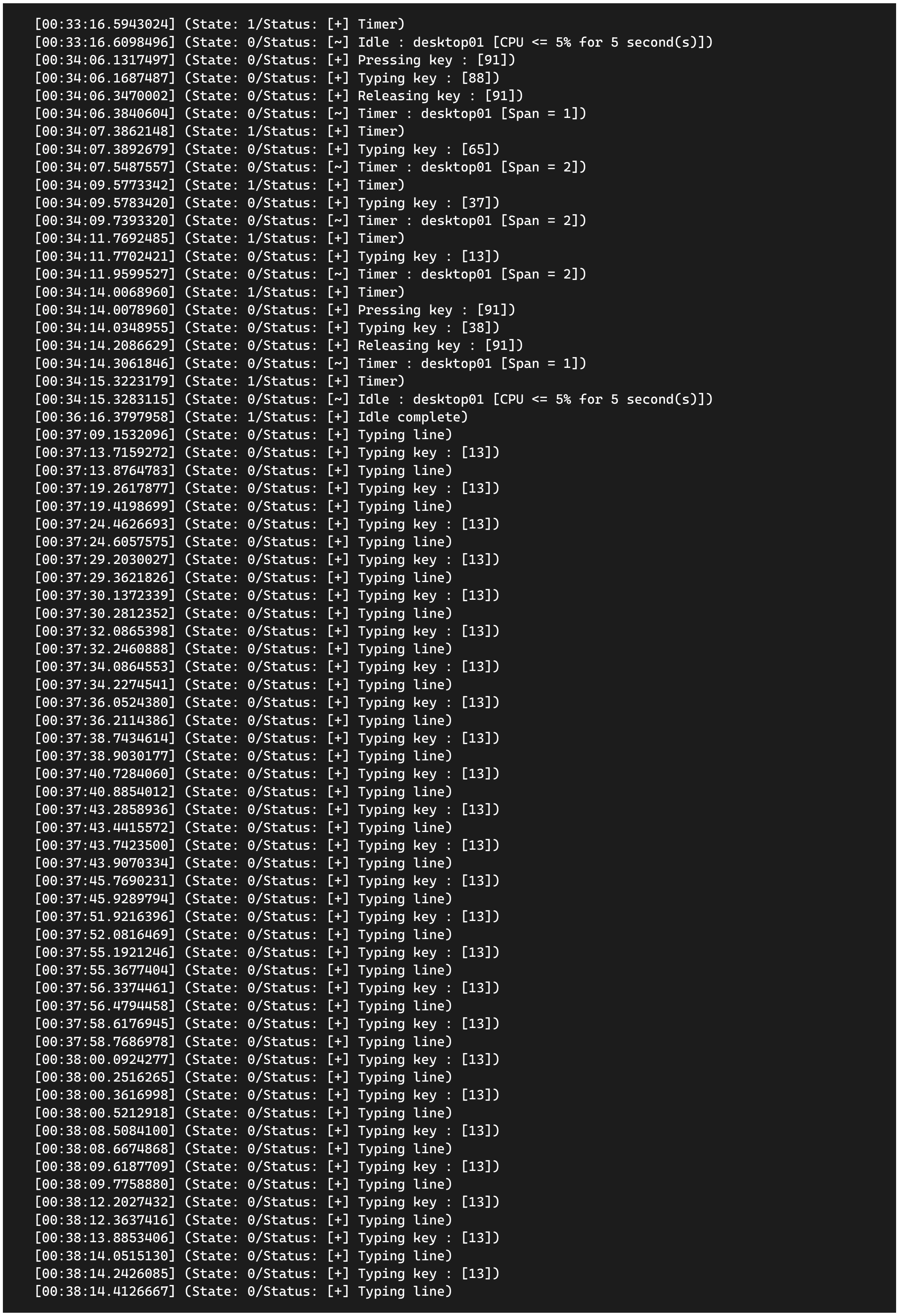
```
    [00:00:00] (State: 0/Status: Running [~] (5/3/2023 12:19:03 PM))
```



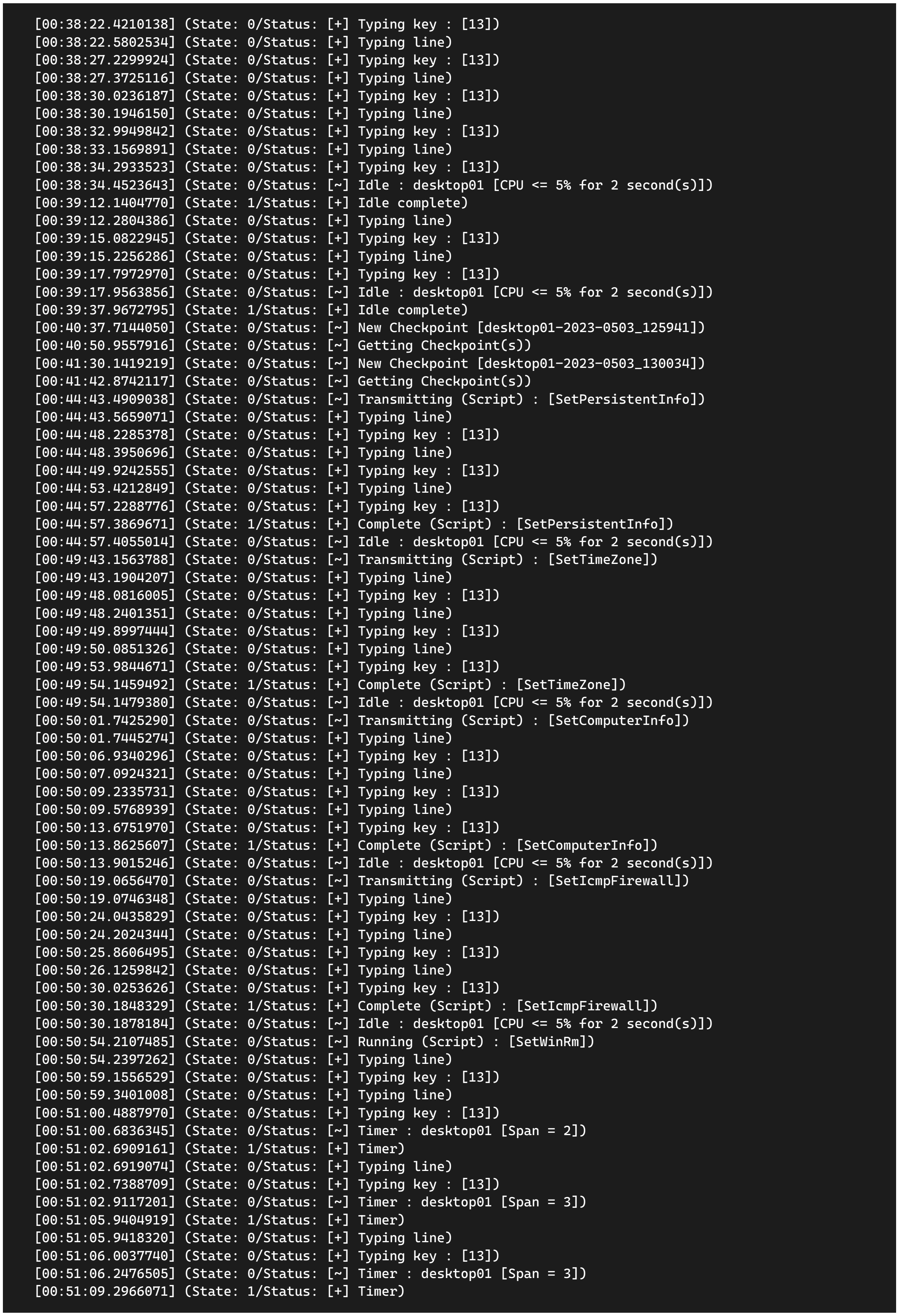
```
[00:00:07.3841138] (State: 0/Status: [~] Creating : desktop01)
[00:00:12.6109497] (State: 0/Status: [~] Getting VmFirmware : desktop01)
[00:00:12.7049496] (State: 0/Status: [~] Setting VmProcessor (Count): [2])
[00:00:15.8945660] (State: 0/Status: [+] Adding VmDvdDrive)
[00:00:16.0985721] (State: 0/Status: [~] Getting VmDvdDrive : desktop01)
[00:00:16.5855686] (State: 0/Status: [~] Setting VmDvdDrive (Path):
    [C:\Images\Win11_22H2_English_x64v1.iso])
[00:00:16.7725739] (State: 0/Status: [~] Setting VmFirmware (Boot order) : [2,0,1])
[00:00:16.7785737] (State: 0/Status: [~] Getting VmFirmware : desktop01)
[00:00:17.1515788] (State: 0/Status: [~] Connecting : desktop01)
[00:00:17.2795688] (State: 1/Status: [~] Starting : desktop01)
[00:00:24.2529934] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:00:26.2841397] (State: 1/Status: [+] Timer)
[00:00:26.2891414] (State: 0/Status: [+] Typing key : [13])
[00:00:26.4631947] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:00:28.5007419] (State: 1/Status: [+] Timer)
[00:00:28.5047348] (State: 0/Status: [+] Typing key : [13])
[00:00:28.6595444] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:00:47.4496685] (State: 1/Status: [+] Idle complete)
[00:00:47.4946661] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:00:49.5058336] (State: 1/Status: [+] Timer)
[00:00:49.5618443] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:01:10.3504013] (State: 1/Status: [+] Idle complete)
[00:01:10.4033799] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:01:12.4218032] (State: 1/Status: [+] Timer)
[00:01:12.4238068] (State: 0/Status: [+] Typing key : [40])
[00:01:12.7234118] (State: 0/Status: [+] Typing key : [40])
[00:01:13.0079494] (State: 0/Status: [+] Typing key : [40])
[00:01:13.3071369] (State: 0/Status: [+] Typing key : [40])
[00:01:13.5786281] (State: 0/Status: [+] Typing key : [40])
[00:01:13.8624640] (State: 0/Status: [+] Typing key : [13])
[00:01:14.0164472] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:01:24.4670987] (State: 1/Status: [+] Idle complete)
[00:01:24.4680820] (State: 0/Status: [+] Typing key : [32])
[00:01:24.6255662] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:01:26.6298603] (State: 1/Status: [+] Timer)
[00:01:26.6922674] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:01:28.7052548] (State: 1/Status: [+] Timer)
[00:01:28.7972521] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:01:30.8144480] (State: 1/Status: [+] Timer)
[00:01:30.8586058] (State: 0/Status: [~] Uptime : desktop01 [Uptime <= 5 second(s)])
[00:05:46.2499684] (State: 1/Status: [+] Uptime complete)
[00:05:46.2519666] (State: 0/Status: [+] Unloading ISO)
[00:05:47.4092066] (State: 0/Status: [~] Timer : desktop01 [Span = 5])
[00:05:52.4433173] (State: 1/Status: [+] Timer)
[00:05:52.4463272] (State: 0/Status: [~] Uptime : desktop01 [Uptime <= 5 second(s)])
[00:13:06.3475841] (State: 1/Status: [+] Uptime complete)
[00:13:06.3495791] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:18:06.5854133] (State: 1/Status: [+] Idle complete)
[00:18:07.4789243] (State: 0/Status: [+] Typing key : [13])
[00:18:07.6474658] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:19:12.3350382] (State: 1/Status: [+] Idle complete)
[00:19:12.3400503] (State: 0/Status: [+] Typing key : [13])
[00:19:12.5110406] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[00:19:13.5270503] (State: 1/Status: [+] Timer)
[00:19:13.5270503] (State: 0/Status: [+] Typing key : [13])
[00:19:13.6860397] (State: 0/Status: [~] Timer : desktop01 [Span = 3])
[00:19:16.6948973] (State: 1/Status: [+] Timer)
[00:19:16.6998986] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:22:19.0120042] (State: 1/Status: [+] Idle complete)
[00:22:19.0139993] (State: 0/Status: [+] Typing key : [9])
[00:22:19.1695800] (State: 0/Status: [+] Typing key : [32])
[00:22:19.3295746] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:22:53.9342856] (State: 1/Status: [+] Idle complete)
[00:22:53.9392922] (State: 0/Status: [+] Typing key : [9])
[00:22:54.1292831] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[00:22:55.1402088] (State: 1/Status: [+] Timer)
[00:22:55.1422095] (State: 0/Status: [+] Typing key : [9])
[00:22:55.2992228] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[00:22:56.3106462] (State: 1/Status: [+] Timer)
[00:22:56.3116353] (State: 0/Status: [+] Typing key : [32])
[00:22:56.4541797] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
```



```
[00:22:57.4628094] (State: 1/Status: [+] Timer)
[00:22:57.4637947] (State: 0/Status: [+] Typing key : [13])
[00:22:57.6223317] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[00:22:58.6532522] (State: 1/Status: [+] Timer)
[00:22:58.6602342] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:23:10.5774055] (State: 1/Status: [+] Idle complete)
[00:23:10.5783793] (State: 0/Status: [+] Typing key : [13])
[00:23:10.7424849] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:23:30.7077545] (State: 1/Status: [+] Idle complete)
[00:23:30.7107575] (State: 0/Status: [+] Typing line)
[00:23:31.8846716] (State: 0/Status: [+] Typing key : [13])
[00:23:32.0427640] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:23:50.9363996] (State: 1/Status: [+] Idle complete)
[00:23:50.9383853] (State: 0/Status: [+] Typing password : [<Password>])
[00:23:52.0018570] (State: 0/Status: [+] Typing key : [13])
[00:23:52.1427840] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:24:23.9614680] (State: 1/Status: [+] Idle complete)
[00:24:24.0214508] (State: 0/Status: [+] Typing key : [13])
[00:24:24.2142025] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:24:43.3073557] (State: 1/Status: [+] Idle complete)
[00:24:43.3133480] (State: 0/Status: [+] Typing key : [13])
[00:24:43.5178698] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:25:08.0194470] (State: 1/Status: [+] Idle complete)
[00:25:08.0234503] (State: 0/Status: [+] Typing text : [<Masked>])
[00:25:08.3214461] (State: 0/Status: [+] Typing key : [9])
[00:25:08.4814480] (State: 0/Status: [+] Typing text : [<Masked>])
[00:25:08.9277292] (State: 0/Status: [+] Typing key : [13])
[00:25:09.0868178] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:25:42.3141227] (State: 1/Status: [+] Idle complete)
[00:25:42.3151217] (State: 0/Status: [+] Typing key : [13])
[00:25:42.4736740] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:25:44.4943948] (State: 1/Status: [+] Timer)
[00:25:44.4954112] (State: 0/Status: [+] Typing key : [13])
[00:25:44.6572970] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:25:46.6815397] (State: 1/Status: [+] Timer)
[00:25:46.6825422] (State: 0/Status: [+] Typing key : [13])
[00:25:46.8395422] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:26:04.9590913] (State: 1/Status: [+] Idle complete)
[00:26:05.3703131] (State: 0/Status: [+] Typing key : [32])
[00:26:05.5426589] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:26:26.5441607] (State: 1/Status: [+] Idle complete)
[00:26:26.7501539] (State: 0/Status: [+] Typing key : [32])
[00:26:26.9942722] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:26:58.1157357] (State: 0/Status: [+] Typing key : [32])
[00:26:58.2918458] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:27:23.9502251] (State: 1/Status: [+] Idle complete)
[00:27:32.7098342] (State: 0/Status: [+] Typing key : [32])
[00:27:32.9229326] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:27:40.3950448] (State: 1/Status: [+] Idle complete)
[00:27:40.6480450] (State: 0/Status: [+] Typing key : [32])
[00:27:40.8305900] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:27:56.6641202] (State: 1/Status: [+] Idle complete)
[00:27:56.9361010] (State: 0/Status: [+] Typing key : [32])
[00:27:57.1451044] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:33:06.9339524] (State: 0/Status: [+] Pressing key : [91])
[00:33:06.9754723] (State: 0/Status: [+] Typing key : [88])
[00:33:07.3604892] (State: 0/Status: [+] Releasing key : [91])
[00:33:07.4144824] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[00:33:08.5703929] (State: 1/Status: [+] Timer)
[00:33:08.5713915] (State: 0/Status: [+] Typing key : [65])
[00:33:08.7293910] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:33:10.7600070] (State: 1/Status: [+] Timer)
[00:33:10.7679938] (State: 0/Status: [+] Typing key : [37])
[00:33:10.9539955] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:33:12.9809489] (State: 1/Status: [+] Timer)
[00:33:13.0059525] (State: 0/Status: [+] Typing key : [13])
[00:33:13.2395658] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:33:15.2903282] (State: 1/Status: [+] Timer)
[00:33:15.2923271] (State: 0/Status: [+] Pressing key : [91])
[00:33:15.3463248] (State: 0/Status: [+] Typing key : [38])
[00:33:15.5254587] (State: 0/Status: [+] Releasing key : [91])
[00:33:15.5854508] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
```

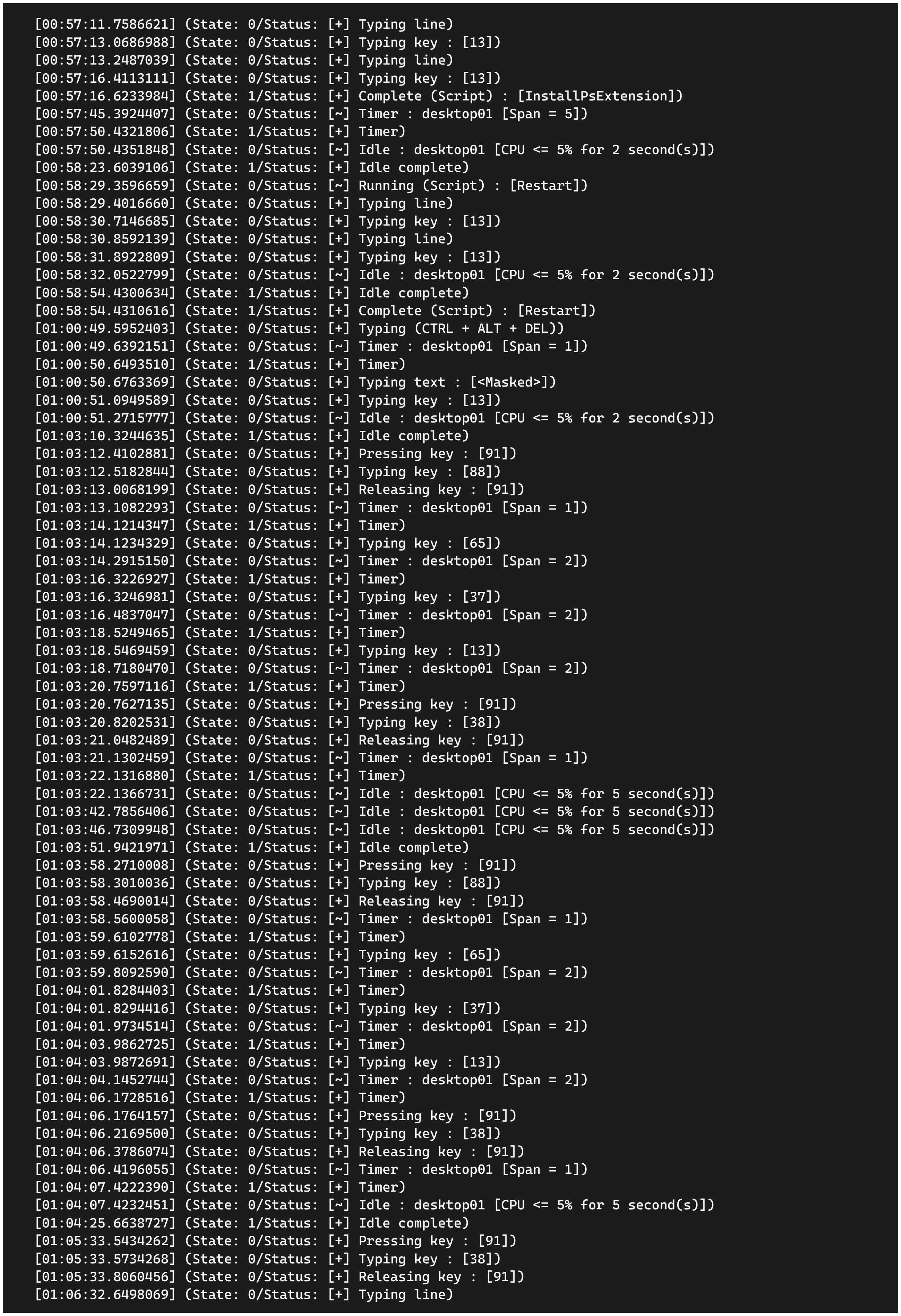


```
[00:33:16.5943024] (State: 1/Status: [+] Timer)
[00:33:16.6098496] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:34:06.1317497] (State: 0/Status: [+] Pressing key : [91])
[00:34:06.1687487] (State: 0/Status: [+] Typing key : [88])
[00:34:06.3470002] (State: 0/Status: [+] Releasing key : [91])
[00:34:06.3840604] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[00:34:07.3862148] (State: 1/Status: [+] Timer)
[00:34:07.3892679] (State: 0/Status: [+] Typing key : [65])
[00:34:07.5487557] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:34:09.5773342] (State: 1/Status: [+] Timer)
[00:34:09.5783420] (State: 0/Status: [+] Typing key : [37])
[00:34:09.7393320] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:34:11.7692485] (State: 1/Status: [+] Timer)
[00:34:11.7702421] (State: 0/Status: [+] Typing key : [13])
[00:34:11.9599527] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:34:14.0068960] (State: 1/Status: [+] Timer)
[00:34:14.0078960] (State: 0/Status: [+] Pressing key : [91])
[00:34:14.0348955] (State: 0/Status: [+] Typing key : [38])
[00:34:14.2086629] (State: 0/Status: [+] Releasing key : [91])
[00:34:14.3061846] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[00:34:15.3223179] (State: 1/Status: [+] Timer)
[00:34:15.3283115] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:36:16.3797958] (State: 1/Status: [+] Idle complete)
[00:37:09.1532096] (State: 0/Status: [+] Typing line)
[00:37:13.7159272] (State: 0/Status: [+] Typing key : [13])
[00:37:13.8764783] (State: 0/Status: [+] Typing line)
[00:37:19.2617877] (State: 0/Status: [+] Typing key : [13])
[00:37:19.4198699] (State: 0/Status: [+] Typing line)
[00:37:24.4626693] (State: 0/Status: [+] Typing key : [13])
[00:37:24.6057575] (State: 0/Status: [+] Typing line)
[00:37:29.2030027] (State: 0/Status: [+] Typing key : [13])
[00:37:29.3621826] (State: 0/Status: [+] Typing line)
[00:37:30.1372339] (State: 0/Status: [+] Typing key : [13])
[00:37:30.2812352] (State: 0/Status: [+] Typing line)
[00:37:32.0865398] (State: 0/Status: [+] Typing key : [13])
[00:37:32.2460888] (State: 0/Status: [+] Typing line)
[00:37:34.0864553] (State: 0/Status: [+] Typing key : [13])
[00:37:34.2274541] (State: 0/Status: [+] Typing line)
[00:37:36.0524380] (State: 0/Status: [+] Typing key : [13])
[00:37:36.2114386] (State: 0/Status: [+] Typing line)
[00:37:38.7434614] (State: 0/Status: [+] Typing key : [13])
[00:37:38.9030177] (State: 0/Status: [+] Typing line)
[00:37:40.7284060] (State: 0/Status: [+] Typing key : [13])
[00:37:40.8854012] (State: 0/Status: [+] Typing line)
[00:37:43.2858936] (State: 0/Status: [+] Typing key : [13])
[00:37:43.4415572] (State: 0/Status: [+] Typing line)
[00:37:43.7423500] (State: 0/Status: [+] Typing key : [13])
[00:37:43.9070334] (State: 0/Status: [+] Typing line)
[00:37:45.7690231] (State: 0/Status: [+] Typing key : [13])
[00:37:45.9289794] (State: 0/Status: [+] Typing line)
[00:37:51.9216396] (State: 0/Status: [+] Typing key : [13])
[00:37:52.0816469] (State: 0/Status: [+] Typing line)
[00:37:55.1921246] (State: 0/Status: [+] Typing key : [13])
[00:37:55.3677404] (State: 0/Status: [+] Typing line)
[00:37:56.3374461] (State: 0/Status: [+] Typing key : [13])
[00:37:56.4794458] (State: 0/Status: [+] Typing line)
[00:37:58.6176945] (State: 0/Status: [+] Typing key : [13])
[00:37:58.7686978] (State: 0/Status: [+] Typing line)
[00:38:00.0924277] (State: 0/Status: [+] Typing key : [13])
[00:38:00.2516265] (State: 0/Status: [+] Typing line)
[00:38:00.3616998] (State: 0/Status: [+] Typing key : [13])
[00:38:00.5212918] (State: 0/Status: [+] Typing line)
[00:38:08.5084100] (State: 0/Status: [+] Typing key : [13])
[00:38:08.6674868] (State: 0/Status: [+] Typing line)
[00:38:09.6187709] (State: 0/Status: [+] Typing key : [13])
[00:38:09.7758880] (State: 0/Status: [+] Typing line)
[00:38:12.2027432] (State: 0/Status: [+] Typing key : [13])
[00:38:12.3637416] (State: 0/Status: [+] Typing line)
[00:38:13.8853406] (State: 0/Status: [+] Typing key : [13])
[00:38:14.0515130] (State: 0/Status: [+] Typing line)
[00:38:14.2426085] (State: 0/Status: [+] Typing key : [13])
[00:38:14.4126667] (State: 0/Status: [+] Typing line)
```



```
[00:38:22.4210138] (State: 0/Status: [+] Typing key : [13])
[00:38:22.5802534] (State: 0/Status: [+] Typing line)
[00:38:27.2299924] (State: 0/Status: [+] Typing key : [13])
[00:38:27.3725116] (State: 0/Status: [+] Typing line)
[00:38:30.0236187] (State: 0/Status: [+] Typing key : [13])
[00:38:30.1946150] (State: 0/Status: [+] Typing line)
[00:38:32.9949842] (State: 0/Status: [+] Typing key : [13])
[00:38:33.1569891] (State: 0/Status: [+] Typing line)
[00:38:34.2933523] (State: 0/Status: [+] Typing key : [13])
[00:38:34.4523643] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:39:12.1404770] (State: 1/Status: [+] Idle complete)
[00:39:12.2804386] (State: 0/Status: [+] Typing line)
[00:39:15.0822945] (State: 0/Status: [+] Typing key : [13])
[00:39:15.2256286] (State: 0/Status: [+] Typing line)
[00:39:17.7972970] (State: 0/Status: [+] Typing key : [13])
[00:39:17.9563856] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:39:37.9672795] (State: 1/Status: [+] Idle complete)
[00:40:37.7144050] (State: 0/Status: [~] New Checkpoint [desktop01-2023-0503_125941])
[00:40:50.9557916] (State: 0/Status: [~] Getting Checkpoint(s))
[00:41:30.1419219] (State: 0/Status: [~] New Checkpoint [desktop01-2023-0503_130034])
[00:41:42.8742117] (State: 0/Status: [~] Getting Checkpoint(s))
[00:44:43.4909038] (State: 0/Status: [~] Transmitting (Script) : [SetPersistentInfo])
[00:44:43.5659071] (State: 0/Status: [+] Typing line)
[00:44:48.2285378] (State: 0/Status: [+] Typing key : [13])
[00:44:48.3950696] (State: 0/Status: [+] Typing line)
[00:44:49.9242555] (State: 0/Status: [+] Typing key : [13])
[00:44:53.4212849] (State: 0/Status: [+] Typing line)
[00:44:57.2288776] (State: 0/Status: [+] Typing key : [13])
[00:44:57.3869671] (State: 1/Status: [+] Complete (Script) : [SetPersistentInfo])
[00:44:57.4055014] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:49:43.1563788] (State: 0/Status: [~] Transmitting (Script) : [SetTimeZone])
[00:49:43.1904207] (State: 0/Status: [+] Typing line)
[00:49:48.0816005] (State: 0/Status: [+] Typing key : [13])
[00:49:48.2401351] (State: 0/Status: [+] Typing line)
[00:49:49.8997444] (State: 0/Status: [+] Typing key : [13])
[00:49:50.0851326] (State: 0/Status: [+] Typing line)
[00:49:53.9844671] (State: 0/Status: [+] Typing key : [13])
[00:49:54.1459492] (State: 1/Status: [+] Complete (Script) : [SetTimeZone])
[00:49:54.1479380] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:50:01.7425290] (State: 0/Status: [~] Transmitting (Script) : [SetComputerInfo])
[00:50:01.7445274] (State: 0/Status: [+] Typing line)
[00:50:06.9340296] (State: 0/Status: [+] Typing key : [13])
[00:50:07.0924321] (State: 0/Status: [+] Typing line)
[00:50:09.2335731] (State: 0/Status: [+] Typing key : [13])
[00:50:09.5768939] (State: 0/Status: [+] Typing line)
[00:50:13.6751970] (State: 0/Status: [+] Typing key : [13])
[00:50:13.8625607] (State: 1/Status: [+] Complete (Script) : [SetComputerInfo])
[00:50:13.9015246] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:50:19.0656470] (State: 0/Status: [~] Transmitting (Script) : [SetIcmpFirewall])
[00:50:19.0746348] (State: 0/Status: [+] Typing line)
[00:50:24.0435829] (State: 0/Status: [+] Typing key : [13])
[00:50:24.2024344] (State: 0/Status: [+] Typing line)
[00:50:25.8606495] (State: 0/Status: [+] Typing key : [13])
[00:50:26.1259842] (State: 0/Status: [+] Typing line)
[00:50:30.0253626] (State: 0/Status: [+] Typing key : [13])
[00:50:30.1848329] (State: 1/Status: [+] Complete (Script) : [SetIcmpFirewall])
[00:50:30.1878184] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:50:54.2107485] (State: 0/Status: [~] Running (Script) : [SetWinRm])
[00:50:54.2397262] (State: 0/Status: [+] Typing line)
[00:50:59.1556529] (State: 0/Status: [+] Typing key : [13])
[00:50:59.3401008] (State: 0/Status: [+] Typing line)
[00:51:00.4887970] (State: 0/Status: [+] Typing key : [13])
[00:51:00.6836345] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[00:51:02.6909161] (State: 1/Status: [+] Timer)
[00:51:02.6919074] (State: 0/Status: [+] Typing line)
[00:51:02.7388709] (State: 0/Status: [+] Typing key : [13])
[00:51:02.9117201] (State: 0/Status: [~] Timer : desktop01 [Span = 3])
[00:51:05.9404919] (State: 1/Status: [+] Timer)
[00:51:05.9418320] (State: 0/Status: [+] Typing line)
[00:51:06.0037740] (State: 0/Status: [+] Typing key : [13])
[00:51:06.2476505] (State: 0/Status: [~] Timer : desktop01 [Span = 3])
[00:51:09.2966071] (State: 1/Status: [+] Timer)
```

```
[00:51:09.2985916] (State: 0/Status: [+] Typing line)
[00:51:13.4986620] (State: 0/Status: [+] Typing key : [13])
[00:51:13.6425409] (State: 0/Status: [~] Timer : desktop01 [Span = 4])
[00:51:17.6889604] (State: 1/Status: [+] Timer)
[00:51:17.6899602] (State: 0/Status: [+] Typing line)
[00:51:17.7549107] (State: 0/Status: [+] Typing key : [13])
[00:51:17.9163520] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:51:22.0884834] (State: 1/Status: [+] Idle complete)
[00:51:22.0894868] (State: 1/Status: [+] Complete (Script) : [SetWinRm])
[00:51:30.5128480] (State: 0/Status: [~] Transmitting (Script) : [SetWinRmFirewall])
[00:51:30.5168326] (State: 0/Status: [+] Typing line)
[00:51:34.5151892] (State: 0/Status: [+] Typing key : [13])
[00:51:34.6730431] (State: 0/Status: [+] Typing line)
[00:51:36.0371433] (State: 0/Status: [+] Typing key : [13])
[00:51:36.2749439] (State: 0/Status: [+] Typing line)
[00:51:39.5203530] (State: 0/Status: [+] Typing key : [13])
[00:51:39.6632268] (State: 1/Status: [+] Complete (Script) : [SetWinRmFirewall])
[00:51:39.6652278] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:51:44.9051801] (State: 1/Status: [+] Idle complete)
[00:51:47.4200881] (State: 0/Status: [~] Transmitting (Script) : [SetRemoteDesktop])
[00:51:47.4230807] (State: 0/Status: [+] Typing line)
[00:51:51.7444422] (State: 0/Status: [+] Typing key : [13])
[00:51:51.8863129] (State: 0/Status: [+] Typing line)
[00:51:53.2626034] (State: 0/Status: [+] Typing key : [13])
[00:51:53.4524372] (State: 0/Status: [+] Typing line)
[00:51:56.8053653] (State: 0/Status: [+] Typing key : [13])
[00:51:56.9473002] (State: 1/Status: [+] Complete (Script) : [SetRemoteDesktop])
[00:51:56.9493007] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:52:02.1785193] (State: 1/Status: [+] Idle complete)
[00:52:09.5158288] (State: 0/Status: [~] Running (Script) : [InstallChoco])
[00:52:09.5178255] (State: 0/Status: [+] Typing line)
[00:52:10.8587584] (State: 0/Status: [+] Typing key : [13])
[00:52:11.0445985] (State: 0/Status: [+] Typing line)
[00:52:15.4002723] (State: 0/Status: [+] Typing key : [13])
[00:52:15.5577406] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:52:46.1536755] (State: 1/Status: [+] Idle complete)
[00:52:46.1566562] (State: 1/Status: [+] Complete (Script) : [InstallChoco])
[00:52:46.1596552] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:52:48.2417930] (State: 1/Status: [+] Idle complete)
[00:52:54.3487029] (State: 0/Status: [~] Running (Script) : [InstallBossMode])
[00:52:54.3547044] (State: 0/Status: [+] Typing line)
[00:52:56.9988128] (State: 0/Status: [+] Typing key : [13])
[00:52:57.1399443] (State: 0/Status: [+] Typing line)
[00:52:58.1460863] (State: 0/Status: [+] Typing key : [13])
[00:52:58.2894750] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:53:09.8421411] (State: 1/Status: [+] Idle complete)
[00:53:09.8431391] (State: 1/Status: [+] Complete (Script) : [InstallBossMode])
[00:53:13.3672036] (State: 0/Status: [~] Transmitting (Script) : [InstallPsExtension])
[00:53:13.3742070] (State: 0/Status: [+] Typing line)
[00:53:17.6789892] (State: 0/Status: [+] Typing key : [13])
[00:53:17.8389801] (State: 0/Status: [+] Typing line)
[00:53:19.1750922] (State: 0/Status: [+] Typing key : [13])
[00:53:19.3480789] (State: 0/Status: [+] Typing line)
[00:53:22.6150206] (State: 0/Status: [+] Typing key : [13])
[00:53:22.7590459] (State: 1/Status: [+] Complete (Script) : [InstallPsExtension])
[00:53:22.7630134] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:53:26.9536610] (State: 1/Status: [+] Idle complete)
[00:54:29.9687666] (State: -1/Status: [!] Exception (Script) : [InstallVsCode] already completed)
[00:54:29.9697700] (State: 0/Status: [~] Running (Script) : [InstallVsCode])
[00:54:29.9927672] (State: 0/Status: [+] Typing line)
[00:54:31.8137774] (State: 0/Status: [+] Typing key : [13])
[00:54:31.9862985] (State: 0/Status: [+] Typing line)
[00:54:33.2612940] (State: 0/Status: [+] Typing key : [13])
[00:54:33.4351572] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:55:15.2038257] (State: 1/Status: [+] Idle complete)
[00:55:15.2048252] (State: 1/Status: [+] Complete (Script) : [InstallVsCode])
[00:55:28.7191051] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[00:56:34.4524802] (State: 1/Status: [+] Idle complete)
[00:57:07.5158171] (State: -1/Status: [!] Exception (Script) : [InstallPsExtension] already completed)
[00:57:07.5168217] (State: 0/Status: [~] Transmitting (Script) : [InstallPsExtension])
[00:57:07.5198202] (State: 0/Status: [+] Typing line)
[00:57:11.5686563] (State: 0/Status: [+] Typing key : [13])
```



```
[00:57:11.7586621] (State: 0/Status: [+] Typing line)
[00:57:13.0686988] (State: 0/Status: [+] Typing key : [13])
[00:57:13.2487039] (State: 0/Status: [+] Typing line)
[00:57:16.4113111] (State: 0/Status: [+] Typing key : [13])
[00:57:16.6233984] (State: 1/Status: [+] Complete (Script) : [InstallPsExtension])
[00:57:45.3924407] (State: 0/Status: [~] Timer : desktop01 [Span = 5])
[00:57:50.4321806] (State: 1/Status: [+] Timer)
[00:57:50.4351848] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:58:23.6039106] (State: 1/Status: [+] Idle complete)
[00:58:29.3596659] (State: 0/Status: [~] Running (Script) : [Restart])
[00:58:29.4016660] (State: 0/Status: [+] Typing line)
[00:58:30.7146685] (State: 0/Status: [+] Typing key : [13])
[00:58:30.8592139] (State: 0/Status: [+] Typing line)
[00:58:31.8922809] (State: 0/Status: [+] Typing key : [13])
[00:58:32.0522799] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[00:58:54.4300634] (State: 1/Status: [+] Idle complete)
[00:58:54.4310616] (State: 1/Status: [+] Complete (Script) : [Restart])
[01:00:49.5952403] (State: 0/Status: [+] Typing (CTRL + ALT + DEL))
[01:00:49.6392151] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[01:00:50.6493510] (State: 1/Status: [+] Timer)
[01:00:50.6763369] (State: 0/Status: [+] Typing text : [<Masked>])
[01:00:51.0949589] (State: 0/Status: [+] Typing key : [13])
[01:00:51.2715777] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 2 second(s)])
[01:03:10.3244635] (State: 1/Status: [+] Idle complete)
[01:03:12.4102881] (State: 0/Status: [+] Pressing key : [91])
[01:03:12.5182844] (State: 0/Status: [+] Typing key : [88])
[01:03:13.0068199] (State: 0/Status: [+] Releasing key : [91])
[01:03:13.1082293] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[01:03:14.1214347] (State: 1/Status: [+] Timer)
[01:03:14.1234329] (State: 0/Status: [+] Typing key : [65])
[01:03:14.2915150] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[01:03:16.3226927] (State: 1/Status: [+] Timer)
[01:03:16.3246981] (State: 0/Status: [+] Typing key : [37])
[01:03:16.4837047] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[01:03:18.5249465] (State: 1/Status: [+] Timer)
[01:03:18.5469459] (State: 0/Status: [+] Typing key : [13])
[01:03:18.7180470] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[01:03:20.7597116] (State: 1/Status: [+] Timer)
[01:03:20.7627135] (State: 0/Status: [+] Pressing key : [91])
[01:03:20.8202531] (State: 0/Status: [+] Typing key : [38])
[01:03:21.0482489] (State: 0/Status: [+] Releasing key : [91])
[01:03:21.1302459] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[01:03:22.1316880] (State: 1/Status: [+] Timer)
[01:03:22.1366731] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[01:03:42.7856406] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[01:03:46.7309948] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[01:03:51.9421971] (State: 1/Status: [+] Idle complete)
[01:03:58.2710008] (State: 0/Status: [+] Pressing key : [91])
[01:03:58.3010036] (State: 0/Status: [+] Typing key : [88])
[01:03:58.4690014] (State: 0/Status: [+] Releasing key : [91])
[01:03:58.5600058] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[01:03:59.6102778] (State: 1/Status: [+] Timer)
[01:03:59.6152616] (State: 0/Status: [+] Typing key : [65])
[01:03:59.8092590] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[01:04:01.8284403] (State: 1/Status: [+] Timer)
[01:04:01.8294416] (State: 0/Status: [+] Typing key : [37])
[01:04:01.9734514] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[01:04:03.9862725] (State: 1/Status: [+] Timer)
[01:04:03.9872691] (State: 0/Status: [+] Typing key : [13])
[01:04:04.1452744] (State: 0/Status: [~] Timer : desktop01 [Span = 2])
[01:04:06.1728516] (State: 1/Status: [+] Timer)
[01:04:06.1764157] (State: 0/Status: [+] Pressing key : [91])
[01:04:06.2169500] (State: 0/Status: [+] Typing key : [38])
[01:04:06.3786074] (State: 0/Status: [+] Releasing key : [91])
[01:04:06.4196055] (State: 0/Status: [~] Timer : desktop01 [Span = 1])
[01:04:07.4222390] (State: 1/Status: [+] Timer)
[01:04:07.4232451] (State: 0/Status: [~] Idle : desktop01 [CPU <= 5% for 5 second(s)])
[01:04:25.6638727] (State: 1/Status: [+] Idle complete)
[01:05:33.5434262] (State: 0/Status: [+] Pressing key : [91])
[01:05:33.5734268] (State: 0/Status: [+] Typing key : [38])
[01:05:33.8060456] (State: 0/Status: [+] Releasing key : [91])
[01:06:32.6498069] (State: 0/Status: [+] Typing line)
```

```
[01:06:36.6427805] (State: 0/Status: [+] Typing key : [13])
[01:06:36.8174125] (State: 0/Status: [+] Typing line)
[01:06:40.5644757] (State: 0/Status: [+] Typing key : [13])
[01:08:12.1184034] (State: 0/Status: [+] Typing line)
[01:08:14.9714390] (State: 0/Status: [+] Typing key : [13])
[01:09:44.7652368] (State: 0/Status: [~] PSSession Token)
[01:09:52.5481914] (State: 0/Status: [~] PSSession Token)
[01:40:41.4373156] (State: 0/Status: [~] Removing : desktop01)
[01:40:41.4973160] (State: 0/Status: [~] State : desktop01 [attempting shutdown])
[01:40:41.5833183] (State: 0/Status: [~] Stopping : desktop01)
[01:41:53.7426447] (State: 0/Status: [~] Vhd  : [C:\VDI\desktop01\desktop01.vhdx])
[01:41:53.7576441] (State: 0/Status: [~] Path : [C:\VDI\desktop01])
[01:41:53.7736448] (State: 0/Status: [~] C:\VDI\desktop01\desktop01\Virtual Machines)
[01:41:53.7796428] (State: 0/Status: [~] C:\VDI\desktop01\desktop01\Snapshots)
[01:41:53.7866417] (State: 0/Status: [~] C:\VDI\desktop01\desktop01)
[01:41:53.7966433] (State: 1/Status: [ ] Removed : C:\VDI\desktop01\desktop01)
[01:41:53.8016457] (State: 100/Status: [+] Dumping console:
    [C:\ProgramData\Secure Digits Plus LLC\Logs\2023-0503_140057-desktop01.log])
[01:41:53.8036648] (State: 100/Status: Complete [+] (5/3/2023 2:00:57 PM), Total: (01:41:53.8036648))
```

```
                                                                                                                        ________/
\______________________________________________________________________________________________________________________/ Output
  Conclusion /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯
```

    So, in this document I was able to unpack and examine all of the aspects of the function [New-VmController].
    Is it done...? Nah.
    Is it perfect...? Nah.
    Will it win a [Nobel Peace Prize] or anything like that...? Probably not.

    But, what it DOES do, is show a way to throw a bunch of variables, functions, classes and types around, into
    a pretty cool little graphical user interface.

```
                                                                                                                    ____________/
\__________________________________________________________________________________________________________________/ Conclusion
________________________________________________________
|¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯|
|                                    Michael C. Cook Sr. |
|                                      Security Engineer |
|                                   Secure Digits Plus LLC |
|________________________________________________________|
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```