VmController [~] 04/17/2023

Introduction

I wanted to take a moment to come up with a brilliant, realistic dialogue between the CEO of [Facebook/Meta], [Mark Zuckerberg], and the dude who made a program back in the day called [Napster], [Sean Parker].

What few people know, is that [Sean Parker] was the man behind [Facebook]'s success.

You might think that it was a bunch of other people, and it might be true to some extent...

However, there were a few key critical players in the game, and it just so happens to be the case, that [Sean Parker] helped [Mark Zuckerberg] take the company to the [next level].

Because, everybody knew.

When it came to [Mark Zuckerberg] asking questions about what direction to take [Facebook] into...?
There's really no arguing that [Sean Parker] was the guy that [Mark] would go to, anytime he needed [really good advice].

/ Introduction

Task At Hand

Consider the following information...

In order to build something [really useful], it has to be [somewhat cool], and it has to [work].
You can't go around doing what the justice system does sometimes, where it is [none] of those things.

So, you have to have a [sense of logic], where [things that make sense] get [implemented].

Things that are [visually appealing], they're [slowly considered] and then [implemented].
Things that are [functionally appealing], they're [slowly considered] and then [implemented].

These are things that many programmers all around the world, will agree with.
Including [Mark Zuckerberg] and [Sean Parker], as well as many others.

Every once in a while, you might be left with a situation where you have to choose between whether you want to [continue developing the idea], OR, maybe you want to [show it off].

When dealing with the [task at hand], it doesn't hurt to [write documents] much like [this one].

Why...?
Because it is [important], in order to [appeal] to [people] that have a [working sense of logic].

You don't want something to just [LOOK] cool, but be really [impractical] or [senseless].
I believe that the [Metaverse] ticks that box repeatedly.

I could imagine [Sean Parker] and [Mark Zuckerberg] having this conversation...

/ Task At Hand

Metaverse (1)

```
[Zuckerberg]: *clicks on [Sean Parker]'s contact*
              *ringing*
[Parker]    : *picks up*
              Hey [Mark Zuckerberg], what's up...?
              Long time no see~!
[Zuckerberg]: Yeh yeh yeh, look... [Sean Parker].
              I need to bend your ear for a minute.
[Parker]    : Whoa, whoa, whoa buddy...
              We've talked about this...
[Zuckerberg]: I know, I know...
              But I'm in a bit of a bind, and I need your help.
[Parker]    : Alright... what's up...?
```

[Zuckerberg]: Trying to build something wicked cool again.
[Parker]   : Buddy, we've both done that about a dozen times over.
[Zuckerberg]: Yeh yeh yeh, I know, but this time is different.
              I've got a lot invested in this cool idea called the [Metaverse].
[Parker]   : *voice trails off* Oh, yeah man.
              It's- gonna be awesome.
[Zuckerberg]: *reponds sharply* It's not going that well, dude.
              Seems like a really big money pit.
[Parker]   : Well, what type of money pit are you talking about...?
              Like, [AMOS-6 satellite] blown up on that [Space X] rocket, money pit...?
[Zuckerberg]: Nah, far worse...
              We're talkin', [billions], dude.
[Parker]   : *quiet*
[Zuckerberg]: ...you there...?
[Parker]   : Yeah.
[Zuckerberg]: Did you hear what I said...?
[Parker]   : *clears throat* Yeah, yeah, I heard what you said, [millions].
[Zuckerberg]: Nah, dude I said [billions].
[Parker]   : *quiet*
[Zuckerberg]: Dude, what the hell...?
[Parker]   : ...what...?
[Zuckerberg]: Stop playing games, you heard me say [billions].
[Parker]   : Yeah.
              Yeah, [Mark Zuckerberg], I did hear you say [billions].
              I'm just... literally blown away by that figure.
              Like, [2 or 3 billion]...?
[Zuckerberg]: Nah, like [36].

   https://www.businessinsider.com/meta-lost-30-billion-on-metaverse-rivals-spent-far-less-2022-10

[Parker]   : *quiet*
[Zuckerberg]: Dude.
              Why do you keep going quiet...?
[Parker]   : Well, [Mark Zuckerberg]...
              That is...
              ...a lot of money you have lost.
[Zuckerberg]: I know, that's why I'm calling you, and asking you for your advice.
[Parker]   : *voice trails off* Well, [Mark Zuckerberg], I am truly flattered...
[Zuckerberg]: Yeah.
              You're doin' it again.
[Parker]   : Look dude, I don't know what you want me to say.
[Zuckerberg]: *sneers* C'mon [Sean Parker].
              I thought you were like the best there is.
[Parker]   : Oh, I'm up there.
              But- times have changed, dude.
              You need to keep up with the times.
              Spending [$36 billion] on a failed venture like the [Metaverse]...?
[Zuckerberg]: ...that's not keeping up with the times...
[Parker]   : Nah.
              You're trying to do something really ambitious, dude.
              It took [Bill Gates] and [Steve Jobs] like (30) years to do what they were
              trying to do...
              points at [Mark Zuckerberg] You need to put this [Metaverse] on the backburner dude.
              Otherwise...?
              You're basically screwing yourself...
              shakes head And I don't want any part of that, at all.
[Zuckerberg]: So you're saying I should just give up...?
[Parker]   : On the [Metaverse]...?
[Zuckerberg]: Yeah.
[Parker]   : Absolutely.
[Zuckerberg]: But- you made [Napster].
              [Napster] was the shit, dude...
              Nobody could even stop you from becoming a true internet sensation.
[Parker]   : I know, I know, you don't have to tell me.
              looks into a picture on his wall from (1998)
              But, that was [then]...
              And this is [now].
              looks out the window It's a whole new ball game now, dude.
[Zuckerberg]: I thought you were like [Duke Nukem], dude.
              Time to kick some ass and chew bubblegum...
[Parker]   : [Mark]...
              I've been outta gum for a real long time.
[Zuckerberg]: [Duke Nukem] wouldn't give up, even if he ran out of gum.

```
              You were never the [just give up] type.
[Parker]    : Nah, look...
              Don't look at it like you're giving up.
              Look at it like you're just putting the hot pan on the back burner...
              ...then you gotta put the slow cooker on, and just wait for a bit.
[Zuckerberg]: So, you're not saying to give up then...?
[Parker]    : Nah, I mean, [temporarily] give up, cause what you're doin' is...
[Zuckerberg]: Difficult.
[Parker]    : Right.
              I mean, you're doin' a bunch of really complicated stuff, and the
              hardware isn't even able to keep up. Let alone the software.
[Zuckerberg]: *opens window blinds, looks out the window* Yeah.
              That's what they keep tellin' me.
[Parker]    : So, if [they] keep tellin' you that, then why aren't you listening...?
[Zuckerberg]: Because I don't wanna wind up like [Steve Jobs]...
              ...building the [Lisa] and then having the company kick me out.
[Parker]    : So, stop spending your time on the [Metaverse].
[Zuckerberg]: But it's my baby, dude.
              How would things have panned out, if you just gave in and never built
              [Napster]...? Hm...?
[Parker]    : Dude, there's no comparison.
              People wanted peer-to-peer mp3's.
              You want to connect everybody and their mother to a virtual world.
              Not a real apples-to-apples comparison.
[Zuckerberg]: *removes fingers from the window blinds* Nah, I suppose you're right.
              Just what the hell am I supposed to do, [Sean Parker]...?
[Parker]    : ...hold off on building the [Metaverse].
[Zuckerberg]: *shakes head* Easier said than done, [Sean Parker].
                                                                                                                  _______________/
\_________________________________________________________________________________________________________________/ Metaverse (1)
```

```powershell
Import-Module FightingEntropy

Function VmXaml
{
    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
        }
    }

    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]>"
        }
    }

    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="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="Extended"/>',
'            <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="50"/>',
'                        <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="120"/>',
'            </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"/>',
'                </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"/>',
'                    </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="150"/>',
'                        <ColumnDefinition Width="100"/>',
'                        <ColumnDefinition Width="150"/>',
'                        <ColumnDefinition Width="*"/>',
'                    </Grid.ColumnDefinitions>',
'                    <Label Grid.Column="0" Content="[Username]:"/>',
'                    <TextBox Grid.Column="1"',
'                             Name="CredentialUsername"/>',
'                    <Label Grid.Column="2" Content="[Password]:"/>',
'                    <PasswordBox Grid.Column="3"',
'                             Name="CredentialPassword"/>',
'                    <Button  Grid.Column="4"',
'                             Name="CredentialGenerate"',
'                             Content="Generate"/>',
'                </Grid>',
'                <Grid Grid.Row="2">',
'                    <Grid.ColumnDefinitions>',
'                        <ColumnDefinition Width="250"/>',
'                        <ColumnDefinition Width="100"/>',
'                        <ColumnDefinition Width="150"/>',
'                        <ColumnDefinition Width="*"/>',
'                    </Grid.ColumnDefinitions>',
'                    <Label Grid.Column="1" Content="[Confirm]:"/>',
'                    <PasswordBox Grid.Column="2"',
'                                 Name="CredentialConfirm"/>',
'                </Grid>',
'            </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}"',
'                                            Width="60"/>',
'                        <DataGridTextColumn Header="Hdd"',
'                                            Binding="{Binding Hdd}"',
'                                            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="100"/>',
'                        <ColumnDefinition Width="120"/>',
'                        <ColumnDefinition Width="120"/>',
'                        <ColumnDefinition Width="*"/>',
'                    </Grid.ColumnDefinitions>',
'                    <Label Grid.Column="0" Content="[Name]:"/>',
'                    <TextBox Grid.Column="1" Name="TemplateName"/>',
'                    <Label Grid.Column="2" Content="[Role]:"/>',
'                    <ComboBox Grid.Column="3" Name="TemplateRole">',
'                        <ComboBoxItem Content="Server"/>',
'                        <ComboBoxItem Content="Client"/>',
'                        <ComboBoxItem Content="Unix"/>',
'                    </ComboBox>',
'                    <Label Grid.Column="4" Content="[Credentials]:"/>',
'                    <TextBox Grid.Column="5"',
'                             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"/>',
'                    </ComboBox>',
'                    <Label Grid.Column="2"',
'                           Content="[Drive/GB]:"',
'                           Style="{StaticResource LabelRed}"/>',
'                    <ComboBox Grid.Column="3"',
'                              Name="TemplateHardDrive"',
'                              SelectedIndex="3">',
'                        <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="100"/>',
```

```
'                            <ColumnDefinition Width="100"/>',
'                        </Grid.ColumnDefinitions>',
'                        <Label    Grid.Column="0" Content="[Name]:"/>',
'                        <TextBox  Grid.Column="1" Name="NodeSwitchName"/>',
'                        <Label    Grid.Column="2" Content="[Type]:"/>',
'                        <ComboBox Grid.Column="3" Name="NodeSwitchType"/>',
'                    </Grid>',
'                </Grid>',
'                <Grid Grid.Row="1" Name="NodeHostPanel" Visibility="Visible">',
'                    <Grid>',
'                        <Grid.RowDefinitions>',
'                            <RowDefinition Height="110"/>',
'                            <RowDefinition Height="40"/>',
'                            <RowDefinition Height="10"/>',
'                            <RowDefinition Height="40"/>',
'                            <RowDefinition Height="*"/>',
'                            <RowDefinition Height="40"/>',
'                        </Grid.RowDefinitions>',
'                        <DataGrid Grid.Row="0" Name="NodeHost">',
'                            <DataGrid.Columns>',
'                                <DataGridTextColumn Header="Index"',
'                                                    Binding="{Binding Index}"',
'                                                    Width="50"/>',
'                                <DataGridTextColumn Header="Name"',
'                                                    Binding="{Binding Name}"',
'                                                    Width="125"/>',
'                                <DataGridTextColumn Header="SwitchName"',
'                                                    Binding="{Binding SwitchName}"',
'                                                    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="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"/>',
'                        <Label Grid.Row="3"',
'                               Content="[Template(s)]: Import template outfile(s)"/>',
'                        <DataGrid Grid.Row="4" Name="NodeTemplate">',
'                            <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}"',
'                                                    Width="60"/>',
'                                <DataGridTextColumn Header="Hdd"',
'                                                    Binding="{Binding Hdd}"',
'                                                    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="5">',
        '                            <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")
    }

    [XamlWindow][VmControllerXaml]::Content
}

Function VmMaster
{
    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>"
        }
    }

    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]>"
        }
    }

    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]>"
        }
    }

    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]>"
        }
    }

    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]>"
        }
    }

    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]>"
        }
    }

    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]>"
        }
    }

    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]>"
        }
    }

    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]>"
        }
    }

    [VmNetworkMaster]::New()
}

Function VmCredential
{
    Enum VmCredentialType
    {
        Setup
        System
        Service
        User
    }

    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
        }
    }

    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   { "Meant for strictly setting up a system"          }
                    System  { "To be used at a system level or for maintenance" }
                    Service { "Allows a service to have access"                 }
                    User    { "Specifically for a user account"                 }
                }

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

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

    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,$Credential)
        }
        [PSCredential] SetCredential([String]$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]"
        }
    }

    [VmCredentialMaster]::New()
}

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

```
            Return $This.Type
        }
    }

    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
        }
    }

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

    Class VmTemplateFile
    {
```

```
        [String]      $Name
        [String]      $Role
        [Object]   $Account
        [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
        [String]     $Image
        VmTemplateFile([Object]$Template,[Object]$Account,[Object]$Network)
        {
            $This.Name      = $Template.Name
            $This.Role      = $Template.Role
            $This.Account   = $Account
            $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
            $This.Image     = $Template.Image
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[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,
        [String]   $Image)
        {
            Return [VmTemplateItem]::New($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,
        [String]   $Image)
        {
            If ($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

            If ([System.IO.File]::Exists($FilePath))
            {
                [Console]::WriteLine("Exported  [+] File: [$FilePath]")
            }
```

```
            Else
            {
                Throw "Something failed... bye."
            }
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmTemplate[Master]>"
        }
    }

    [VmTemplateMaster]::New()
}

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

```
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]>"
    }
}

Class VmNodeTemplate
{
    [UInt32]     $Index
    [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
    [String]     $Image
    VmNodeTemplate([UInt32]$Index,[Object]$File)
    {
        $Item           = Import-CliXml -Path $File.Fullname
        $This.Index     = $Index
        $This.Name      = $Item.Name
        $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     = $Item.Image
    }
    [Object] VmNodeDhcp([Object]$Dhcp)
```

```
        {
            Return [VmNodeDhcp]::New($Dhcp)
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Template]>"
        }
    }

    Class VmNodeItem
    {
        [UInt32]      $Index
        [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.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] VmByteSize([String]$Name,[UInt64]$Bytes)
        {
            Return [VmByteSize]::New($Name,$Bytes)
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Item]>"
        }
    }

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

    Class VmNodeHost
    {
        [UInt32]      $Index
        [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.Name       = $Node.Name
            $This.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]>"
        }
    }

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

            $This.Path = $Path
        }
        Clear([String]$Slot)
        {
            Switch -Regex ($Slot)
            {
                "Switch"   { $This.Switch   = @( ) }
                "Host"     { $This.Host     = @( ) }
                "Template" { $This.Template = @( ) }
            }
        }
        [Object] VmNodeSwitch([UInt32]$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[]] GetVmSwitch()
        {
            Return Get-VmSwitch
        }
        [Object[]] GetVm()
        {
```

```
        Return Get-Vm
    }
    [Object[]] GetTemplate()
    {
        Return Get-ChildItem $This.Path | ? Extension -eq .fex
    }
    AddTemplate([Object]$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,$Node)
    }
    Refresh([String]$Type)
    {
        If ($Type -notin "Switch","Host","Template")
        {
            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)
                    }
                }
            }
        }
    }
    Refresh()
    {
        $This.Clear("Switch")
        $This.Clear("Host")
        $This.Clear("Template")

        # Switch
        ForEach ($Item in $This.GetVmSwitch())
        {
            $This.AddSwitch($Item)
        }

        # Host
        ForEach ($Item in $This.GetVm())
        {
            $This.AddHost($Item)
        }
```

```
            # Templates
            If ($This.Path)
            {
                ForEach ($Item in $This.GetTemplate())
                {
                    $This.AddTemplate($Item)
                }
            }
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Master]>"
        }
    }

    [VmNodeMaster]::New()
}

Class GridProperty
{
    [String]  $Name
    [Object] $Value
    GridProperty([Object]$Property)
    {
        $This.Name  = $Property.Name
        $This.Value = $Property.Value -join ", "
    }
}

Class ValidateFlag
{
    [UInt32] $Index
    [String] $Name
    [UInt32] $Status
    ValidateFlag([UInt32]$Index,[String]$Name)
    {
        $This.Index  = $Index
        $This.Name   = $Name
        $This.SetStatus(0)
    }
    SetStatus([UInt32]$Status)
    {
        $This.Status = $Status
    }
}

Class VmMasterController
{
    [Object]     $Module
    [Object]    $Console
    [Object]       $Xaml
    [Object]     $Master
    [Object] $Credential
    [Object]   $Template
    [Object]       $Node
    [Object]       $Flag
    VmMasterController()
    {
        $This.Module      = $This.GetFEModule()
        $This.Module.Mode = 0
        $This.Xaml        = $This.VmXaml()
        $This.Master      = $This.VmMaster()
        $This.Credential  = $This.VmCredential()
        $This.Template    = $This.VmTemplate()
        $This.Node        = $This.VmNode()
        $This.Flag        = @( )

        ForEach ($Name in "MasterPath",
                          "MasterDomain",
                          "MasterNetBios",
                          "CredentialUsername",
```

```
                        "CredentialPassword",
                        "CredentialConfirm",
                        "TemplatePath",
                        "TemplateImagePath",
                        "NodeTemplatePath")
        {
            $This.Flag += $This.ValidateFlag($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()
    {
        Return Get-FEModule -Mode 1
    }
    [Object] VmXaml()
    {
        $This.Update(0,"Getting [~] VmXaml")
        Return VmXaml
    }
    [Object] VmMaster()
    {
        $This.Update(0,"Getting [~] VmMaster")
        Return VmMaster
    }
    [Object] VmCredential()
    {
```

```
        $This.Update(0,"Getting [~] VmCredential")
        Return VmCredential
    }
    [Object] VmTemplate()
    {
        $This.Update(0,"Getting [~] VmTemplate")
        Return VmTemplate
    }
    [Object] VmNode()
    {
        $This.Update(0,"Getting [~] VmNode")
        Return VmNode
    }
    [Object] ValidateFlag([UInt32]$Index,[String]$Name)
    {
        Return [ValidateFlag]::New($Index,$Name)
    }
    [Object] GridProperty([Object]$Property)
    {
        Return [GridProperty]::New($Property)
    }
    SetNetwork([UInt32]$Index)
    {
        $This.Update(0,"Setting [~] Network")
        $This.Master.SetNetwork($Index)

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

        $This.Update(0,"Setting [~] Dhcp")
        $This.Master.Network.SetDhcp()
    }
    PingSweep([Object[]]$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
        $Item        = $This.Flag | ? Name -eq CredentialUsername
        $Item.Status = [UInt32]($Username -ne "" -and $Username -notin $This.Credential.Output)
    }
```

```
        CheckPassword()
        {
            $Password    = $This.Xaml.IO.CredentialPassword.Password
            $Item        = $This.Flag | ? Name -eq CredentialPassword
            $Item.Status = [UInt32]($Password -ne "")
        }
        CheckConfirm()
        {
            $Password    = [Regex]::Escape($This.Xaml.IO.CredentialPassword.Password)
            $Confirm     = [Regex]::Escape($This.Xaml.IO.CredentialConfirm.Password)
            $Item        = $This.Flag | ? Name -eq CredentialConfirm
            $Item.Status = [UInt32]($Password -ne "" -and $Password -eq $Confirm)
        }
        ToggleCredentialCreate()
        {
            $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)
        }
        ToggleTemplateCreate()
        {
            $C = 0
            ForEach ($Item in $This.Flag | ? Name -match "^Template")
            {
                If ($Item.Status -eq 1)
                {
                    $C ++
                }
            }

            $This.Xaml.IO.TemplateCreate.IsEnabled = $C -eq 2
        }
        CheckPath([String]$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()
    }
    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()
    }
    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 | % { $This.GridProperty($_) }
    }
    [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 | % { $This.GridProperty($_) }
    }
    SetInitialState()
    {
        $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

        # 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

        # 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")
    }
    Invoke()
    {
        Try
        {
            $This.Xaml.Invoke()
        }
        Catch
        {
            $This.Module.Write(1,"Failed [!] Either the user cancelled or the dialog failed")
        }
    }
    StageXaml()
    {
        $Ctrl = $This

        <#


             0 MasterConfig       DataGrid System.Windows.Controls.DataGrid Items.Count:1
             1 MasterPath         TextBox  System.Windows.Controls.TextBox: C:\FileVm
             2 MasterPathIcon     Image    System.Windows.Controls.Image
             3 MasterPathBrowse   Button   System.Windows.Controls.Button: Browse
             4 MasterDomain       TextBox  System.Windows.Controls.TextBox: securedigitsplus.com
             5 MasterDomainIcon   Image    System.Windows.Controls.Image
             6 MasterNetBios      TextBox  System.Windows.Controls.TextBox: secured
             7 MasterNetBiosIcon  Image    System.Windows.Controls.Image
             8 MasterCreate       Button   System.Windows.Controls.Button: Create
             9 MasterConfigOutput DataGrid System.Windows.Controls.DataGrid Items.Count:22
            10 MasterBase         DataGrid System.Windows.Controls.DataGrid Items.Count:10
            11 MasterRange        DataGrid System.Windows.Controls.DataGrid Items.Count:1
            12 MasterHosts        DataGrid System.Windows.Controls.DataGrid Items.Count:256
            13 MasterDhcp         DataGrid System.Windows.Controls.DataGrid Items.Count:7
        #>

        $Ctrl.Reset($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
          ___//--\\

---\\_____________________________________________________________________________________________________//
--\\__//

-------------------------------------------------------------------------------------------------------
 ----
            14 CredentialOutput      DataGrid    System.Windows.Controls.DataGrid Items.Count:2
            15 CredentialCreate      Button      System.Windows.Controls.Button: Create
            16 CredentialRemove      Button      System.Windows.Controls.Button: Remove
            17 CredentialType        ComboBox    System.Windows.Controls.ComboBox Items.Count:4
            18 CredentialDescription DataGrid    System.Windows.Controls.DataGrid Items.Count:0
            19 CredentialUsername    TextBox     System.Windows.Controls.TextBox: T
            20 CredentialPassword    PasswordBox System.Windows.Controls.PasswordBox
            21 CredentialGenerate    Button      System.Windows.Controls.Button: Generate
            22 CredentialConfirm     PasswordBox System.Windows.Controls.PasswordBox
        #>

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

        $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.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.Xaml.IO.CredentialRemove.IsEnabled  = $Ctrl.Xaml.IO.CredentialOutput.SelectedIndex -ne
-1
        })

        $Ctrl.Xaml.IO.CredentialRemove.Add_Click(
        {
            Switch ($Ctrl.Xaml.IO.CredentialOutput.Items.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.Credential.Output)
            $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)

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

            $Ctrl.Xaml.IO.TemplateCredentialCount.Text = $Ctrl.Credential.Output.Count
            $Ctrl.Xaml.IO.CredentialUsername.Text      = ""
            $Ctrl.Xaml.IO.CredentialPassword.Password  = ""
            $Ctrl.Xaml.IO.CredentialConfirm.Password   = ""
        })

        $Ctrl.Reset($Ctrl.Xaml.IO.CredentialOutput,$Ctrl.Credential.Output)

        <#
            ____
 ___________________________________________________________________________________________________


 //--\\__//-------------------------------------------------------------------------------------------
----\\___
          \\__//--- Template [~] Panel
           ___//--\\

---\\_____________________________________________________________________________________________//
--\\__//

-------------------------------------------------------------------------------------------------
 ____
            23 TemplateOutput          DataGrid System.Windows.Controls.DataGrid Items.Count:0
            24 TemplateCreate          Button   System.Windows.Controls.Button: Create
            25 TemplateRemove          Button   System.Windows.Controls.Button: Remove
            26 TemplateExport          Button   System.Windows.Controls.Button: Export
            27 TemplateName            TextBox  System.Windows.Controls.TextBox
            28 TemplateRole            ComboBox System.Windows.Controls.ComboBox Items.Count:3
            29 TemplateCredentialCount TextBox  System.Windows.Controls.TextBox
            30 TemplatePath            TextBox  System.Windows.Controls.TextBox: <Select a path>
            31 TemplatePathIcon        Image    System.Windows.Controls.Image
            32 TemplatePathBrowse      Button   System.Windows.Controls.Button: Browse
```

```
    33 TemplateMemory          ComboBox System.Windows.Controls.ComboBox Items.Count:2
    34 TemplateHardDrive       ComboBox System.Windows.Controls.ComboBox Items.Count:4
    35 TemplateGeneration      ComboBox System.Windows.Controls.ComboBox Items.Count:2
    36 TemplateCore            ComboBox System.Windows.Controls.ComboBox Items.Count:4
    37 TemplateSwitch          ComboBox System.Windows.Controls.ComboBox Items.Count:2
    38 TemplateImagePath       TextBox  System.Windows.Controls.TextBox: <Select an image>
    39 TemplateImagePathIcon   Image    System.Windows.Controls.Image
    40 TemplateImagePathBrowse Button   System.Windows.Controls.Button: Browse
#>

$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")
})

ForEach ($Item in "TemplateCreate","TemplateRemove","TemplateExport")
{
    $Ctrl.Xaml.Get($Item).IsEnabled = 0
}

$Ctrl.Xaml.IO.TemplateCreate.Add_Click(
{
    If ($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")
    }

    Else
    {
        $Ctrl.Template.Add($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,
                           $Ctrl.Xaml.IO.TemplateImagePath.Text)

        $Ctrl.Reset($Ctrl.Xaml.IO.TemplateOutput,$Ctrl.Template.Output)

        $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.Xaml.IO.TemplateExport.IsEnabled = $Ctrl.Xaml.IO.TemplateOutput.Items.Count -gt 0
```

```
            $Ctrl.Xaml.IO.TemplateRemove.IsEnabled = $Ctrl.Xaml.IO.TemplateOutput.SelectedIndex -ne -1
        })

        $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.Template.Output)
        })

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

        <#
            ____
 ------------------------------------------------------------------------------------------------------


 //--\\__//------------------------------------------------------------------------------------------
----\\___
          \\__//--- Node [~] Panel
           ___//--\\

---\\_____________________________________________________________________________________________//
--\\__//

------------------------------------------------------------------------------------------------
 ----
            41 NodeSlot             ComboBox System.Windows.Controls.ComboBox Items.Count:2
            42 NodeSwitchPanel      Grid     System.Windows.Controls.Grid
            43 NodeSwitch           DataGrid System.Windows.Controls.DataGrid Items.Count:2
            44 NodeSwitchCreate     Button   System.Windows.Controls.Button: Create
            45 NodeSwitchRemove     Button   System.Windows.Controls.Button: Remove
            46 NodeSwitchUpdate     Button   System.Windows.Controls.Button: Update
            47 NodeSwitchName       TextBox  System.Windows.Controls.TextBox
            48 NodeSwitchType       ComboBox System.Windows.Controls.ComboBox Items.Count:0
            49 NodeHostPanel        Grid     System.Windows.Controls.Grid
            50 NodeHost             DataGrid System.Windows.Controls.DataGrid Items.Count:2
            51 NodeHostCreate       Button   System.Windows.Controls.Button: Create
            52 NodeHostRemove       Button   System.Windows.Controls.Button: Remove
            53 NodeHostUpdate       Button   System.Windows.Controls.Button: Update
            54 NodeTemplate         DataGrid System.Windows.Controls.DataGrid Items.Count:0
            55 NodeTemplateImport   Button   System.Windows.Controls.Button: Import
            56 NodeTemplatePath     TextBox  System.Windows.Controls.TextBox
            57 NodeTemplatePathIcon Image    System.Windows.Controls.Image
        #>

        $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]
        })

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

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

        $Ctrl.Xaml.IO.NodeHostUpdate.Add_Click(
        {
            $Ctrl.Node.Refresh("Host")
            $Ctrl.Reset($Ctrl.Xaml.IO.NodeHost,$Ctrl.Node.Host)
```

```
        })

        $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("Template")

            $Ctrl.Reset($Ctrl.Xaml.IO.NodeTemplate,$Ctrl.Node.Template)
        })

        $Ctrl.SetInitialState()
    }
}

$Ctrl = [VmMasterController]::New()
$Ctrl.StageXaml()
$Ctrl.Xaml.Get("MasterPath").Text            = "C:\FileVm"
$Ctrl.Xaml.Get("MasterDomain").Text          = "securedigitsplus.com"
$Ctrl.Xaml.Get("MasterNetBios").Text         = "secured"
$Ctrl.Xaml.Get("MasterConfig").SelectedIndex = 0
$Ctrl.Invoke()
```