```
 ____    __________________________________________________________________________________________________
//¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
\\__//¯¯¯ Get-FEDCPromo [~] (01/03/2023)                                                        ___//¯¯\\
 ¯¯¯\\__________________________________________________________________________________________//¯¯\\__//
     ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
```

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

Greetings,

Today I will be covering a utility that has been taking me quite a long time to (develop/redevelop) over the last couple of years or so. Actually, this utility has been going back to some point in 2019... though in various stages and whatnot.

This utility is actually pretty advanced and complicated, because it orchestrates the process of controlling the Active Directory Domain Controller promotion process, and... it requires a pretty thorough understanding of so many things that some people may not even know or understand.

This includes an understanding of networking, domain prerequisites, server administration, IP classes, roles and windows features, and then to top all of it off, I'm able to orchestrate all of these things from a Xaml/PowerShell based graphical user interface that is able to test the various functions and then log it all to files for testing, validation, and review... as well as continuation after necessary reboots.

Simply put, this function is ******* complicated.

It's not quite “balancing a broomhandle in a hurricane” level of complicated, but, there were a TEAM of guys that accomplished that very complicated process...
...of conceptualizing a rocket that can land itself...
...so that it can be (reused + refueled)...
...and reduce the cost of sending things into space to (1/10) of what it used to cost...
...when NASA just threw a lot of money at making rockets that weren't meant to be reused.

NASA has historically made the best rockets that were so expensive and never meant to be reused, that they have their own Hall of Fame. Sounds stupid, right...? Well, think again.

Below, I will paste the function wrapper. The function wrapper is effectively the thing that goes on the outside of the classes below that do all of the work.

Then, I will paste each individual class below.

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

 //==================================================================================================\\
//  Module     : [FightingEntropy()][2022.12.0]                                                       \\
\\  Date       : 2023-01-02 17:25:30                                                                  //
 \\==================================================================================================//

    FileName   : Get-FEDCPromo.ps1
    Solution   : [FightingEntropy()][2022.12.0]
    Purpose    : For the promotion of a FightingEntropy (ADDS/Various) Domain Controller
    Author     : Michael C. Cook Sr.
    Contact    : @mcc85s
    Primary    : @mcc85s
    Created    : 2022-12-14
    Modified   : 2023-01-02
    Demo       : N/A
    Version    : 0.0.0 - () - Finalized functional version 1
    TODO       : [~] Test each level and functionality across each mode
                 [~] Nbt scanner (modes 1..3)

.Example
#>
Function Get-FEDCPromo
{
```

```
    [CmdLetBinding()]Param(
    [Parameter()][UInt32]$Mode=0,
    [Parameter()][Hashtable]$InputObject)

    # Check for server operating system
    If (Get-CimInstance Win32_OperatingSystem | ? Caption -notmatch Server)
    {
        Throw "Must use Windows Server operating system"
    }

    <# Insert classes here #>

    $Ctrl = [FEDCPromoController]::New($Mode)

    $Ctrl.Xaml.Invoke()

}
```

This function isn't complete, and I'm going to intentionally leave out some functionality in this specific document, because when it is ready, I'll be able to demonstrate all of these various functions in action, in a demonstration video that shoestrings everything working, back-to-back.

Class [XamlProperty] / Introduction

This class is covered in other portions of the module, it can effectively be used to locate a specific control for the Xaml from the [XamlWindow] object, or from the main [FEDCPromoController] class below.

I haven't capitalized on how this property works quite yet, though to be clear, that isn't because I don't have plans to do so in the future.

```
    # (1/4) [Xaml.Property]
    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] / Class [XamlProperty]

This class right here, is the main class to instantiate a block of Xaml. It uses the above class to organize the controls that are able to be interacted with by [PowerShell], as well as the [System.Xml.XmlNodeReader] and [System.Windows.Markup.XamlReader] classes available via the [Windows Presentation Framework].

```
    # (2/4) [Xaml.Window]
    Class XamlWindow
    {
        Hidden [Object]        $XAML
        Hidden [Object]         $XML
        [String[]]            $Names
```

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

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

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

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

_____/ Class [XamlWindow]

Class [FEDCFoundXaml] /----------\
/----------

This particular chunk of Xaml is meant to show a list of found domain controllers that are found via the
NBT scanning process. Here's how that process actually works...

1) all active network adapters are polled
2) any active network adapter with an IP address is then queried for any potential hosts on that given network
3) for each adapter, a number of calculations are performed to determine the RANGE of potential host IP addresses
4) performs a ping sweep to look for any ACTIVE nodes that respond
5) uses the command nbtstat to remotely scan the nodes who responded to a ping request in the ping sweep
6) parses the output for each response, and then shows whether they are domain controllers <1B> or <1C>
7) all of those things that meet all of those criteria, they populate the datagrid in this Xaml object.

Complicated, right...?
That's right.

```
# (3/4) [Xaml.FEDCFound]
Class FEDCFoundXaml
{
    Static [String] $Content = @(
    '<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" ',
    '        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" ',
    '        Title="[FightingEntropy]://Domain Controller Found"',
    '        Width="550"',
    '        Height="260"',
    '        HorizontalAlignment="Center"',
    '        Topmost="True"',
    '        ResizeMode="NoResize"',
    '        Icon="C:\ProgramData\Secure Digits Plus LLC\FightingEntropy\2022.12.0\Graphics\icon.ico"',
    '        WindowStartupLocation="CenterScreen">',
    '    <Window.Resources>',
    '        <Style TargetType="GroupBox">',
    '            <Setter Property="Margin" Value="10"/>',
    '            <Setter Property="Padding" Value="10"/>',
    '            <Setter Property="TextBlock.TextAlignment" Value="Center"/>',
    '            <Setter Property="Template">',
    '                <Setter.Value>',
    '                    <ControlTemplate TargetType="GroupBox">',
    '                        <Border CornerRadius="10"',
    '                                Background="White"',
    '                                BorderBrush="Black"',
    '                                BorderThickness="3">',
    '                            <ContentPresenter x:Name="ContentPresenter"',
    '                                              ContentTemplate="{TemplateBinding ContentTemplate}"',
    '                                              Margin="5"/>',
    '                        </Border>',
    '                    </ControlTemplate>',
    '                </Setter.Value>',
    '            </Setter>',
    '        </Style>',
    '        <Style TargetType="Button">',
    '            <Setter Property="Margin" Value="5"/>',
    '            <Setter Property="Padding" Value="5"/>',
    '            <Setter Property="Height" Value="30"/>',
    '            <Setter Property="FontWeight" Value="Semibold"/>',
    '            <Setter Property="FontSize" Value="12"/>',
    '            <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="DataGridCell">',
    '            <Setter Property="TextBlock.TextAlignment" Value="Left" />',
    '        </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="BorderBrush" Value="Black"/>',
        '            <Style.Triggers>',
        '                <Trigger Property="AlternationIndex" Value="0">',
        '                    <Setter Property="Background" Value="White"/>',
        '                </Trigger>',
        '                <Trigger Property="AlternationIndex" Value="1">',
        '                    <Setter Property="Background" Value="#FFD6FFFB"/>',
        '                </Trigger>',
        '            </Style.Triggers>',
        '        </Style>',
        '    </Window.Resources>',
        '    <Grid>',
        '        <Grid.Background>',
        '            <ImageBrush Stretch="None"',
        '                        ImageSource="C:\ProgramData\Secure Digits Plus
LLC\FightingEntropy\2022.12.0\Graphics\background.jpg"/>',
        '        </Grid.Background>',
        '        <GroupBox>',
        '            <Grid Margin="5">',
        '                <Grid.RowDefinitions>',
        '                    <RowDefinition Height="*"/>',
        '                    <RowDefinition Height="50"/>',
        '                </Grid.RowDefinitions>',
        '                <DataGrid Grid.Row="0" Grid.Column="0" Name="DomainControllers">',
        '                    <DataGrid.Columns>',
        '                        <DataGridTextColumn Header="Address"',
        '                                            Width="140"',
        '                                            Binding="{Binding IPAddress}"/>',
        '                        <DataGridTextColumn Header="Hostname"',
        '                                            Width="200"',
        '                                            Binding="{Binding HostName}"/>',
        '                        <DataGridTextColumn Header="NetBIOS"',
        '                                            Width="140"',
        '                                            Binding="{Binding NetBIOS}"/>',
        '                    </DataGrid.Columns>',
        '                </DataGrid>',
        '                <Grid Grid.Row="1">',
        '                    <Grid.ColumnDefinitions>',
        '                        <ColumnDefinition Width="*"/>',
        '                        <ColumnDefinition Width="*"/>',
        '                    </Grid.ColumnDefinitions>',
        '                    <Button Grid.Row="1"',
        '                            Grid.Column="0"',
        '                            Name="Ok"',
        '                            Content="Ok" />',
        '                    <Button Grid.Row="1"',
        '                            Grid.Column="1"',
        '                            Content="Cancel"',
        '                            Name="Cancel"/>',
        '                </Grid>',
        '            </Grid>',
        '        </GroupBox>',
        '    </Grid>',
        '</Window>' -join "`n")
    }
```

Class [FEDCPromoXaml]

Class [FEDCFoundXaml]

This is the Xaml object for the controller. It used to be a rather large window, until I just recently compacted the layout of this function. The result is that it is much smaller than it was before, but- now things are spread out across a number of tabs.

However, I added a number of new features, for instance, anything preventing the function from enabling the START or TEST buttons, they're in the summary datagrid. There are additional things I'm going to add, such as the above chunk of Xaml will eventually be included in this Xaml object below.

I'll do that at some point when I have to shore everything up for the next release.
That's what I've been doing the last (2) months or so...

```
# (4/4) [Xaml.FEDCPromo]
Class FEDCPromoXaml
{
    Static [String] $Content = @(
    '<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"',
    '        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"',
    '        Title="[FightingEntropy]://Domain Controller Promotion"',
    '        Width="550"',
    '        Height="450"',
    '        Topmost="True"',
    '        ResizeMode="NoResize"',
    '        Icon="C:\ProgramData\Secure Digits Plus LLC\FightingEntropy\2022.12.0\Graphics\icon.ico"',
    '        HorizontalAlignment="Center"',
    '        WindowStartupLocation="CenterScreen">',
    '    <Window.Resources>',
    '        <Style x:Key="DropShadow">',
    '            <Setter Property="TextBlock.Effect">',
    '                <Setter.Value>',
    '                    <DropShadowEffect ShadowDepth="1"/>',
    '                </Setter.Value>',
    '            </Setter>',
    '        </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>',
    '        <Style TargetType="CheckBox">',
    '            <Setter Property="VerticalContentAlignment" Value="Center"/>',
    '            <Setter Property="Height" Value="24"/>',
    '            <Setter Property="Margin" Value="5"/>',
    '        </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="2"',
'                                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>',
'                            <Trigger Property="IsEnabled" Value="False">',
'                                <Setter TargetName="Border"',
'                                        Property="Background"',
'                                        Value="#6F6F6F"/>',
'                                <Setter Property="Foreground"',
'                                        Value="#9F9F9F"/>',
'                            </Trigger>',
'                        </ControlTemplate.Triggers>',
'                    </ControlTemplate>',
'                </Setter.Value>',
'            </Setter>',
'        </Style>',
'        <Style TargetType="Button">',
'            <Setter Property="Margin" Value="5"/>',
'            <Setter Property="Padding" Value="5"/>',
'            <Setter Property="Height" Value="30"/>',
'            <Setter Property="FontWeight" Value="Semibold"/>',
'            <Setter Property="FontSize" Value="12"/>',
'            <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="ComboBox">',
'            <Setter Property="Height" Value="24"/>',
'            <Setter Property="Margin" Value="5"/>',
'            <Setter Property="FontSize" Value="12"/>',
'            <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="Margin" Value="5"/>',
'            <Setter Property="Padding" Value="5"/>',
```

```
'            <Setter Property="BorderThickness" Value="2"/>',
'            <Setter Property="BorderBrush" Value="Black"/>',
'            <Setter Property="Foreground" Value="Black"/>',
'        </Style>',
'        <Style TargetType="TextBox" x:Key="Block">',
'            <Setter Property="Margin" Value="5"/>',
'            <Setter Property="Padding" Value="5"/>',
'            <Setter Property="FontFamily" Value="Consolas"/>',
'            <Setter Property="Height" Value="180"/>',
'            <Setter Property="FontSize" Value="10"/>',
'            <Setter Property="FontWeight" Value="Normal"/>',
'            <Setter Property="AcceptsReturn" Value="True"/>',
'            <Setter Property="VerticalAlignment" Value="Top"/>',
'            <Setter Property="TextAlignment" Value="Left"/>',
'            <Setter Property="VerticalContentAlignment" Value="Top"/>',
'            <Setter Property="VerticalScrollBarVisibility" Value="Visible"/>',
'            <Setter Property="TextBlock.Effect">',
'                <Setter.Value>',
'                    <DropShadowEffect ShadowDepth="1"/>',
'                </Setter.Value>',
'            </Setter>',
'        </Style>',
'        <Style TargetType="DataGrid">',
'            <Setter Property="Margin" Value="5"/>',
'            <Setter Property="AutoGenerateColumns" Value="False"/>',
'            <Setter Property="AlternationCount" Value="3"/>',
'            <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="BorderBrush" Value="Black"/>',
'        </Style>',
'        <Style TargetType="DataGridColumnHeader">',
'            <Setter Property="FontSize"   Value="8"/>',
'            <Setter Property="FontWeight" Value="Medium"/>',
'            <Setter Property="Margin" Value="2"/>',
'            <Setter Property="Padding" Value="2"/>',
'        </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>',
'    <Grid Margin="5">',
'        <Grid.Resources>',
'            <Style TargetType="Grid">',
'                <Setter Property="Background" Value="LightYellow"/>',
'            </Style>',
'        </Grid.Resources>',
'        <Grid.RowDefinitions>',
'            <RowDefinition Height="40"/>',
'            <RowDefinition Height="*"/>',
'            <RowDefinition Height="40"/>',
'        </Grid.RowDefinitions>',
'        <Grid Grid.Row="0">',
'            <Grid.ColumnDefinitions>',
'                <ColumnDefinition Width="100"/>',
'                <ColumnDefinition Width="70"/>',
'                <ColumnDefinition Width="*"/>',
'            </Grid.ColumnDefinitions>',
'            <Label Grid.Column="0"',
'                   Content="[Command]:"',
'                   HorizontalContentAlignment="Center"/>',
'            <ComboBox Grid.Column="1" Name="CommandSlot">',
'                <ComboBoxItem Content="Forest"/>',
'                <ComboBoxItem Content="Tree"/>',
'                <ComboBoxItem Content="Child"/>',
'                <ComboBoxItem Content="Clone"/>',
'            </ComboBox>',
'            <DataGrid Grid.Column="2"',
'                      Margin="10"',
'                      Name="Command"',
'                      HeadersVisibility="None">',
'                <DataGrid.Columns>',
'                    <DataGridTextColumn Header="Description" ',
'                                        Width="*" ',
'                                        Binding="{Binding Description}"/>',
'                </DataGrid.Columns>',
'            </DataGrid>',
'        </Grid>',
'        <TabControl Grid.Row="1">',
'            <TabItem Header="Mode">',
'                <Grid>',
'                    <Grid.RowDefinitions>',
'                        <RowDefinition Height="30"/>',
'                        <RowDefinition Height="50"/>',
'                        <RowDefinition Height="10"/>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="10"/>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="10"/>',
'                        <RowDefinition Height="40"/>',
'                    </Grid.RowDefinitions>',
```

```
'                    <DataGrid Grid.Row="0"',
'                              Name="OperatingSystemCaption"',
'                              HeadersVisibility="None">',
'                        <DataGrid.Columns>',
'                            <DataGridTextColumn Header="Caption"',
'                                                Width="*"',
'                                                Binding="{Binding Caption}"/>',
'                        </DataGrid.Columns>',
'                    </DataGrid>',
'                    <DataGrid Grid.Row="1" Name="OperatingSystemExtension">',
'                        <DataGrid.Columns>',
'                            <DataGridTextColumn Header="Version"',
'                                                Width="80"',
'                                                Binding="{Binding Version}"/>',
'                            <DataGridTextColumn Header="Build"',
'                                                Width="50"',
'                                                Binding="{Binding Build}"/>',
'                            <DataGridTextColumn Header="Serial"',
'                                                Width="*"',
'                                                Binding="{Binding Serial}"/>',
'                            <DataGridTextColumn Header="Language"',
'                                                Width="50"',
'                                                Binding="{Binding Language}"/>',
'                            <DataGridTextColumn Header="Product"',
'                                                Width="50"',
'                                                Binding="{Binding Product}"/>',
'                            <DataGridTextColumn Header="Type"',
'                                                Width="50"',
'                                                Binding="{Binding Type}"/>',
'                        </DataGrid.Columns>',
'                    </DataGrid>',
'                    <Grid Grid.Row="3" Name="ForestModeBox">',
'                        <Grid.ColumnDefinitions>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="120"/>',
'                            <ColumnDefinition Width="50"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="25"/>',
'                        </Grid.ColumnDefinitions>',
'                        <Label Grid.Column="1"',
'                               Content="Forest Mode"',
'                               Style="{StaticResource LabelGray}"/>',
'                        <ComboBox Grid.Column="2"',
'                                  Name="ForestMode"',
'                                  SelectedIndex="0">',
'                            <ComboBox.ItemTemplate>',
'                                <DataTemplate>',
'                                    <TextBlock Text="{Binding Index}"',
'                                               IsEnabled="{Binding Enabled}"/>',
'                                </DataTemplate>',
'                            </ComboBox.ItemTemplate>',
'                        </ComboBox>',
'                        <DataGrid Grid.Column="3"',
'                                  Name="ForestModeExtension"',
'                                  HeadersVisibility="None"',
'                                  Margin="10">',
'                            <DataGrid.Columns>',
'                                <DataGridTextColumn Header="Name"',
'                                                    Binding="{Binding Name}"',
'                                                    Width="100"/>',
'                                <DataGridTextColumn Header="DisplayName"',
'                                                    Binding="{Binding DisplayName}"',
'                                                    Width="*"/>',
'                            </DataGrid.Columns>',
'                        </DataGrid>',
'                    </Grid>',
'                    <Grid Grid.Row="4" Name="DomainModeBox">',
'                        <Grid.ColumnDefinitions>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="120"/>',
'                            <ColumnDefinition Width="50"/>',
'                            <ColumnDefinition Width="*"/>',
```

```
'                        <ColumnDefinition Width="25"/>',
'                    </Grid.ColumnDefinitions>',
'                    <Label Grid.Column="1"',
'                           Content="Domain Mode"',
'                           Style="{StaticResource LabelGray}"/>',
'                    <ComboBox Grid.Column="2"',
'                              Name="DomainMode"',
'                              SelectedIndex="0">',
'                        <ComboBox.ItemTemplate>',
'                            <DataTemplate>',
'                                <TextBlock Text="{Binding Index}"',
'                                           IsEnabled="{Binding Enabled}"/>',
'                            </DataTemplate>',
'                        </ComboBox.ItemTemplate>',
'                    </ComboBox>',
'                    <DataGrid Grid.Column="3"',
'                              Name="DomainModeExtension"',
'                              HeadersVisibility="None"',
'                              Margin="10">',
'                        <DataGrid.Columns>',
'                            <DataGridTextColumn Header="Name"',
'                                                Binding="{Binding Name}"',
'                                                Width="100"/>',
'                            <DataGridTextColumn Header="DisplayName"',
'                                                Binding="{Binding DisplayName}"',
'                                                Width="*"/>',
'                        </DataGrid.Columns>',
'                    </DataGrid>',
'                </Grid>',
'                <Grid Grid.Row="6" Name="ParentDomainNameBox">',
'                    <Grid.ColumnDefinitions>',
'                        <ColumnDefinition Width="25"/>',
'                        <ColumnDefinition Width="120"/>',
'                        <ColumnDefinition Width="25"/>',
'                        <ColumnDefinition Width="*"/>',
'                        <ColumnDefinition Width="25"/>',
'                    </Grid.ColumnDefinitions>',
'                    <Label   Grid.Column="1"',
'                             Content="Parent Domain"',
'                             Style="{StaticResource LabelGray}"/>',
'                    <Image   Grid.Column="2"',
'                             Name="ParentDomainNameIcon"/>',
'                    <TextBox Grid.Column="3" ',
'                             Name="ParentDomainName"/>',
'                </Grid>',
'                <Grid Grid.Row="8" Name="ReplicationSourceDCBox">',
'                    <Grid.ColumnDefinitions>',
'                        <ColumnDefinition Width="25"/>',
'                        <ColumnDefinition Width="120"/>',
'                        <ColumnDefinition Width="*"/>',
'                        <ColumnDefinition Width="25"/>',
'                    </Grid.ColumnDefinitions>',
'                    <Label    Grid.Column="1"',
'                              Content="Replication DC"',
'                              Style="{StaticResource LabelGray}"/>',
'                    <ComboBox Grid.Column="2" Name="ReplicationSourceDC"/>',
'                </Grid>',
'            </Grid>',
'        </TabItem>',
'        <TabItem Header="Features">',
'            <Grid>',
'                <Grid.RowDefinitions>',
'                    <RowDefinition Height="40"/>',
'                    <RowDefinition Height="*"/>',
'                </Grid.RowDefinitions>',
'                <Label    Grid.Row="0"',
'                          Content="[Windows Server features to be installed]:"/>',
'                <DataGrid Grid.Row="1"',
'                          Name="Feature">',
'                    <DataGrid.Columns>',
'                        <DataGridTextColumn Header="Type"',
'                                            Width="60"',
```

```
'                                           Binding="{Binding Type}"',
'                                           CanUserSort="True"',
'                                           IsReadOnly="True"/>',
'                            <DataGridTextColumn Header="Name"',
'                                           Width="*"',
'                                           Binding="{Binding Name}"',
'                                           CanUserSort="True"',
'                                           IsReadOnly="True"',
'                                           FontWeight="Bold"/>',
'                            <DataGridTemplateColumn Header="Install" Width="40">',
'                                <DataGridTemplateColumn.CellTemplate>',
'                                    <DataTemplate>',
'                                        <CheckBox IsEnabled="{Binding Enable}"',
'                                                  IsChecked="{Binding Install}"',
'                                                  Margin="0"',
'                                                  Height="18"',
'                                                  HorizontalAlignment="Left"/>',
'                                    </DataTemplate>',
'                                </DataGridTemplateColumn.CellTemplate>',
'                            </DataGridTemplateColumn>',
'                        </DataGrid.Columns>',
'                    </DataGrid>',
'                </Grid>',
'            </TabItem>',
'            <TabItem Header="Roles/Paths">',
'                <Grid>',
'                    <Grid.RowDefinitions>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="40"/>',
'                        <RowDefinition Height="40"/>',
'                    </Grid.RowDefinitions>',
'                    <Label Grid.Row="0" Content="[Domain controller roles]"/>',
'                    <Grid Grid.Row="1">',
'                        <Grid.ColumnDefinitions>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="32"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="32"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="25"/>',
'                        </Grid.ColumnDefinitions>',
'                        <CheckBox Grid.Column="1"',
'                                  Name="InstallDNS"/>',
'                        <Label    Grid.Column="2"',
'                                  Content="Install DNS"',
'                                  Style="{StaticResource LabelRed}"/>',
'                        <CheckBox Grid.Column="3"',
'                                  Name="NoGlobalCatalog"/>',
'                        <Label    Grid.Column="4"',
'                                  Content="No Global Catalog"',
'                                  Style="{StaticResource LabelRed}"/>',
'                    </Grid>',
'                    <Grid Grid.Row="2">',
'                        <Grid.ColumnDefinitions>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="32"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="32"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="25"/>',
'                        </Grid.ColumnDefinitions>',
'                        <CheckBox Grid.Column="1"',
'                                  Name="CreateDNSDelegation"/>',
'                        <Label    Grid.Column="2"',
'                                  Content="Create DNS Delegation"',
'                                  Style="{StaticResource LabelRed}"/>',
'                        <CheckBox Grid.Column="3"',
'                                  Name="CriticalReplicationOnly"/>',
```

```
        '                        <Label    Grid.Column="4"',
        '                                  Content="Critical Replication Only"',
        '                                  Style="{StaticResource LabelRed}"/>',
        '                    </Grid>',
        '                    <Label Grid.Row="3"',
        '                           Content="[Active Directory partition target paths]"/>',
        '                    <Grid Grid.Row="4">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="120"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Button   Grid.Column="1"',
        '                                  Name="DatabaseBrowse"',
        '                                  Content="Database"/>',
        '                        <TextBox  Grid.Column="2"',
        '                                  Name="DatabasePath"/>',
        '                    </Grid>',
        '                    <Grid Grid.Row="5">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="120"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Button   Grid.Column="1"',
        '                                  Name="SysvolBrowse"',
        '                                  Content="SysVol"/>',
        '                        <TextBox  Grid.Column="2"',
        '                                  Name="SysvolPath"/>',
        '                    </Grid>',
        '                    <Grid Grid.Row="6">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="120"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Button  Grid.Column="1"',
        '                                 Name="LogBrowse"',
        '                                 Content="Log"/>',
        '                        <TextBox Grid.Column="2"',
        '                                 Name="LogPath"/>',
        '                    </Grid>',
        '                </Grid>',
        '            </TabItem>',
        '            <TabItem Header="Names">',
        '                <Grid>',
        '                    <Grid.RowDefinitions>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="20"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                    </Grid.RowDefinitions>',
        '                    <Label Grid.Row="0"',
        '                           Content="[Necessary fields vary by command selection]"/>',
        '                    <Grid   Grid.Row="2" Name="DomainNameBox">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Label   Grid.Column="1"',
        '                                 Content="Domain"',
        '                                 Style="{StaticResource LabelGray}"/>',
        '                        <Image   Grid.Column="2"',
```

```
'                                Name="DomainNameIcon"/>',
'                        <TextBox Grid.Column="3"',
'                                 Name="DomainName"/>',
'                    </Grid>',
'                    <Grid Grid.Row="3" Name="NewDomainNameBox">',
'                        <Grid.ColumnDefinitions>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="100"/>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="25"/>',
'                        </Grid.ColumnDefinitions>',
'                        <Label   Grid.Column="1"',
'                                 Content="New Domain"',
'                                 Style="{StaticResource LabelGray}"/>',
'                        <Image   Grid.Column="2"',
'                                 Name="NewDomainNameIcon"/>',
'                        <TextBox Grid.Column="3"',
'                                 Name="NewDomainName"/>',
'                    </Grid>',
'                    <Grid Grid.Row="4" Name="DomainNetBiosNameBox">',
'                        <Grid.ColumnDefinitions>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="100"/>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="25"/>',
'                        </Grid.ColumnDefinitions>',
'                        <Label   Grid.Column="1"',
'                                 Content="NetBIOS"',
'                                 Style="{StaticResource LabelGray}"/>',
'                        <Image   Grid.Column="2"',
'                                 Name="DomainNetBIOSNameIcon"/>',
'                        <TextBox Grid.Column="3"',
'                                 Name="DomainNetBIOSName"/>',
'                    </Grid>',
'                    <Grid Grid.Row="5" Name="NewDomainNetBiosNameBox">',
'                        <Grid.ColumnDefinitions>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="100"/>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="25"/>',
'                        </Grid.ColumnDefinitions>',
'                        <Label   Grid.Column="1"',
'                                 Content="New NetBIOS"',
'                                 Style="{StaticResource LabelGray}"/>',
'                        <Image   Grid.Column="2"',
'                                 Name="NewDomainNetBIOSNameIcon"/>',
'                        <TextBox Grid.Column="3"',
'                                 Name="NewDomainNetBIOSName"/>',
'                    </Grid>',
'                    <Grid Grid.Row="6" Name="SiteNameBox">',
'                        <Grid.ColumnDefinitions>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="100"/>',
'                            <ColumnDefinition Width="25"/>',
'                            <ColumnDefinition Width="*"/>',
'                            <ColumnDefinition Width="25"/>',
'                        </Grid.ColumnDefinitions>',
'                        <Label    Grid.Column="1"',
'                                  Content="Site Name"',
'                                  Style="{StaticResource LabelGray}"/>',
'                        <Image    Grid.Column="2"',
'                                  Name="SiteNameIcon"/>',
'                        <ComboBox Grid.Column="3"',
'                                  Name="SiteName"/>',
'                    </Grid>',
'                </Grid>',
'            </TabItem>',
'            <TabItem Header="Credential">',
'                <Grid>',
```

```
        '                    <Grid.RowDefinitions>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="20"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="20"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="20"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="10"/>',
        '                    </Grid.RowDefinitions>',
        '                    <Label Grid.Row="0"',
        '                           Content="[Active Directory promotion credential]"/>',
        '                    <Grid  Grid.Row="2" Name="CredentialBox">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Button  Grid.Column="1"',
        '                                 Content="Credential"',
        '                                 Name="CredentialButton"/>',
        '                        <Image   Grid.Column="2"',
        '                                 Name="CredentialIcon"/>',
        '                        <TextBox Grid.Column="3"',
        '                                 Name="Credential"/>',
        '                    </Grid>',
        '                    <Label Grid.Row="4"',
        '                           Content="[(DSRM/Domain Services Restore Mode) Key]"/>',
        '                    <Grid Grid.Row="6">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Label Grid.Column="1"',
        '                               Content="Password"',
        '                               Style="{StaticResource LabelGray}"/>',
        '                        <Image Grid.Column="2"',
        '                               Name="SafeModeAdministratorPasswordIcon"/>',
        '                        <PasswordBox Grid.Column="3"',
        '                                     Name="SafeModeAdministratorPassword"/>',
        '                    </Grid>',
        '                    <Grid Grid.Row="7">',
        '                        <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="100"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                            <ColumnDefinition Width="25"/>',
        '                        </Grid.ColumnDefinitions>',
        '                        <Label Grid.Column="1"',
        '                               Content="Confirm"',
        '                               Style="{StaticResource LabelGray}"/>',
        '                        <Image Grid.Column="2"',
        '                               Name="ConfirmIcon"/>',
        '                        <PasswordBox Grid.Column="3"',
        '                                     Name="Confirm"/>',
        '                    </Grid>',
        '                </Grid>',
        '            </TabItem>',
        '            <TabItem Header="Summary">',
        '                <Grid>',
        '                    <Grid.RowDefinitions>',
        '                        <RowDefinition Height="40"/>',
        '                        <RowDefinition Height="*"/>',
        '                    </Grid.RowDefinitions>',
        '                    <Label Grid.Row="0" Content="[Issues preventing promotion]"/>',
```

```
        '                    <DataGrid Grid.Row="1" Name="Summary">',
        '                        <DataGrid.Columns>',
        '                            <DataGridTextColumn Header="Name"',
        '                                                Binding="{Binding Name}"',
        '                                                Width="150"/>',
        '                            <DataGridTextColumn Header="Reason"',
        '                                                Binding="{Binding Reason}"',
        '                                                Width="*"/>',
        '                        </DataGrid.Columns>',
        '                    </DataGrid>',
        '                </Grid>',
        '            </TabItem>',
        '        </TabControl>',
        '        <Grid Grid.Row="2">',
        '            <Grid.ColumnDefinitions>',
        '                <ColumnDefinition Width="*"/>',
        '                <ColumnDefinition Width="*"/>',
        '                <ColumnDefinition Width="*"/>',
        '            </Grid.ColumnDefinitions>',
        '            <Button Grid.Column="0"',
        '                    Name="Start"',
        '                    Content="Start"/>',
        '            <Button Grid.Column="1"',
        '                    Name="Test"',
        '                    Content="Test"/>',
        '            <Button Grid.Column="2"',
        '                    Name="Cancel"',
        '                    Content="Cancel"/>',
        '        </Grid>',
        '    </Grid>',
        '</Window>' -join "`n")
    }
```

Class [FEDCPromoXaml]

Class [FeatureItem]

This class right here allows each desired Windows Optional feature to be queried and then populated here. This effectively allows the Xaml objects to be changed in response to the state of the properties of this particular object, for each desired feature.

```
    # (1/2) [Feature.Item]
    Class FeatureItem
    {
        [UInt32]   $Index
        [String]    $Type
        [String]    $Name
        [Object]   $State
        [UInt32]  $Enable
        [UInt32] $Install
        FeatureItem([UInt32]$Index,[String]$Type,[String]$Name)
        {
            $This.Index   = $Index
            $This.Type    = $Type
            $This.Name    = $Name
        }
        Set([Object]$Feature)
        {
            $This.State   = [UInt32]$Feature.Installed
            $This.Enable  = $This.State -ne 1
            $This.Install = $This.State -ne 1
        }
        Toggle()
        {
            $This.Install         = !$This.Install
        }
        [String] ToString()
        {
            Return $This.Name
```

```
        }
    }
```

_____/
_____/ Class [FeatureItem]
Class [FeatureController] /_____\
/_____

This allows a template of Windows Features to be used, and then they are displayed in the GUI.

```
    # (2/2) [Feature.Controller]
    Class FeatureController
    {
        [String]     $Name
        [Object]   $Output
        FeatureController()
        {
            $This.Name       = "Feature"
            $This.Clear()
            $This.Stage()
        }
        Clear()
        {
            $This.Output     = @( )
            $This.Output     = $This.GetTemplate()
        }
        Stage()
        {
            ForEach ($Feature in Get-WindowsFeature | ? Name -in $This.Output.Name)
            {
                $Item        = $This.Output | ? Name -eq $Feature.Name
                $Item.Set($Feature)
            }
        }
        [Object] GetTemplate()
        {
            $Out = @( )
            ForEach ($Type in "Main","WDS","IIS","Veridian")
            {
                $Slot        = Switch ($Type)
                {
                    Main     { $This.Main()     }
                    WDS      { $This.WDS()      }
                    IIS      { $This.IIS()      }
                    Veridian { $This.Veridian() }
                }

                ForEach ($Item in $Slot)
                {
                    $Out += $This.FeatureItem($Out.Count,$Type,$Item)
                }
            }

            Return $Out
        }
        [String[]] Main()
        {
            $Out = "AD-Domain-Services DHCP DNS GPMC ! !-AD-AdminCenter !-AD-PowerShell "+
            "!-AD-Tools !-ADDS !-ADDS-Tools !-DHCP !-DNS-Server !-Role-Tools"

            Return $Out -Replace "!","RSAT" -Split " "
        }
        [String[]] WDS()
        {
            $Out = "! !-AdminPack !-Deployment !-Transport"

            Return $Out -Replace "!","WDS" -Split " "
        }
        [String[]] IIS()
```

```
        {
            $Out = "BITS BITS-IIS-Ext DSC-Service FS-SMBBW ManagementOData Net-Framework"+
            "-45-ASPNet Net-WCF-HTTP-Activation45 RSAT-BITS-Server WAS WAS-Config-APIs W"+
            "AS-Process-Model WebDAV-Redirector !HTTP-Errors !HTTP-Logging !HTTP-Redirec"+
            "t !HTTP-Tracing !App-Dev !AppInit !Asp-Net45 !Basic-Auth !Common-Http !Cust"+
            "om-Logging !DAV-Publishing !Default-Doc !Digest-Auth !Dir-Browsing !Filteri"+
            "ng !Health !Includes !Log-Libraries !Metabase !Mgmt-Console !Net-Ext45 !Per"+
            "formance !Request-Monitor !Security !Stat-Compression !Static-Content !Url-"+
            "Auth !WebServer !Windows-Auth !ISAPI-Ext !ISAPI-Filter !Server WindowsPower"+
            "ShellWebAccess"

            Return $Out -Replace "!","Web-" -Split " "
        }
        [String[]] Veridian()
        {
            $Out = "! RSAT-!-Tools !-Tools !-PowerShell"

            Return $Out -Replace "!","Hyper-V" -Split " "
        }
        [Object] FeatureItem([UInt32]$Index,[String]$Type,[String]$Name)
        {
            Return [FeatureItem]::New($Index,$Type,$Name)
        }
        [String] ToString()
        {
            Return "<FEDCPromo.FeatureController>"
        }
    }
```

Enum [WindowsServerType] / Class [FeatureController]

This displays the available Windows Server types for <ForestMode> and <DomainMode>.

```
    # (1/3) [Windows.Server.Type]
    Enum WindowsServerType
    {
        Win2K
        Win2003
        Win2008
        Win2008R2
        Win2012
        Win2012R2
        Win2016
        Win2019
        Win2022
    }
```

Class [WindowsServerItem] / Enum [WindowsServerType]

Allows the above enumeration type to be indexed via the ComboBox in the Xaml, as well as displayed in the GUI.

```
    # (2/3) [Windows.Server.Item]
    Class WindowsServerItem
    {
        [UInt32]       $Index
        [String]        $Name
        [String] $DisplayName
        [UInt32]      $Enable = 1
        WindowsServerItem([String]$Name)
        {
            $This.Index       = [UInt32][WindowsServerType]::$Name
```

```
            $This.Name        = $Name
        }
        [String] ToString()
        {
            Return $This.DisplayName
        }
    }
```

_____/
_____/ Class [WindowsServerItem]
Class [WindowsServerList] /_____\
/_____

Creates a bunch of stuff for the Xaml to interact with on the backend, in relation to the Windows Server items.

```
    # (3/3) [Windows.Server.List]
    Class WindowsServerList
    {
        [String]     $Name
        [UInt32] $Selected
        [Object]   $Output
        WindowsServerList([String]$Name)
        {
            $This.Name   = $Name
            $This.Output = @( )

            ForEach ($Name in [System.Enum]::GetNames([WindowsServerType]))
            {
                $This.Add($Name)
            }
        }
        [Object] WindowsServerItem([String]$Name)
        {
            Return [WindowsServerItem]::New($Name)
        }
        Add([String]$Name)
        {
            $Item             = $This.WindowsServerItem($Name)
            $Item.DisplayName = Switch ($Item.Index)
            {
                0 { "Windows Server 2000"    }
                1 { "Windows Server 2003"    }
                2 { "Windows Server 2008"    }
                3 { "Windows Server 2008 R2" }
                4 { "Windows Server 2012"    }
                5 { "Windows Server 2012 R2" }
                6 { "Windows Server 2016"    }
                7 { "Windows Server 2019"    }
                8 { "Windows Server 2022"    }
            }

            $This.Output     += $Item
        }
        SetMin([UInt32]$Index)
        {
            If ($Index -gt $This.Output.Count)
            {
                Throw "Invalid entry"
            }

            ForEach ($Item in $This.Output)
            {
                $Item.Enable = [UInt32]($Item.Index -ge $Index)
            }
        }
        [Object] Current()
        {
            Return $This.Output[$This.Selected]
        }
        [String] ToString()
```

```
        {
            Return "({0}) <FEDCPromo.WindowsServerList[{1}]>" -f $This.Output.Count, $This.Name
        }
    }
```

_____/ Class [WindowsServerList] _____/ Enum [ProfileSlotType] /_____\

In the GUI, these items are either ENABLED or DISABLED depending on the selected mode.
The selected mode is effectively what the desired command and testing criteria will be.

```
# (1/10) [Profile.Slot.Type]
Enum ProfileSlotType
{
    ForestMode
    DomainMode
    ReplicationSourceDC
    SiteName
    ParentDomainName
    DomainName
    DomainNetBIOSName
    NewDomainName
    NewDomainNetBIOSName
}
```

_____/ Enum [ProfileSlotType] _____/ Class [ProfileSlotItem] /_____\

The controls in the Xaml, and desired command mode, they ALL have to have (control + validation) criteria.
That's what this object here is doing.

```
# (2/10) [Profile.Slot.Item]
Class ProfileSlotItem
{
    [UInt32]      $Index
    [String]       $Name
    [String]       $Type
    [String]   $Property
    [UInt32]  $IsEnabled
    [Object]      $Value
    [UInt32]      $Check
    [String]     $Reason
    ProfileSlotItem([UInt32]$Index,[String]$Name,[Bool]$IsEnabled)
    {
        $This.Index     = $Index
        $This.Name      = $Name
        $This.Type      = @("TextBox","ComboBox"     )[[UInt32]($Index -in 0..3)]
        $This.Property  = @("Text"   ,"SelectedIndex")[[UInt32]($Index -in 0..3)]
        $This.IsEnabled = $IsEnabled
        $This.Check     = 0
    }
    Set([Object]$Value)
    {
        $This.Value     = $Value
    }
    Validate([String]$Reason)
    {
        If ($This.Type -eq "TextBox")
        {
            $This.Check  = [UInt32]($Reason -eq "[+] Passed")
            $This.Reason = $Reason
        }
    }
```

```
        [String] ToString()
        {
            Return "<FEDCPromo.ProfileSlotItem>"
        }
    }
```

_____/
_____/ Class [ProfileSlotItem]
Class [ProfileSlotList] /_____\
/_____

This builds the current profile based on the selected command, as well as setting the default values for the Xaml object.

```
    # (3/10) [Profile.Slot.List]
    Class ProfileSlotList
    {
        [UInt32]   $Mode
        [String]   $Name
        [Object] $Output
        ProfileSlotList([UInt32]$Mode)
        {
            $This.Mode   = $Mode
            $This.Name   = "Slot"
            $This.Stage()
        }
        Clear()
        {
            $This.Output = @( )
        }
        Stage()
        {
            $This.Clear()

            ForEach ($Name in [System.Enum]::GetNames([ProfileSlotType]))
            {
                $IsEnabled = @(Switch ($Name)
                {
                    ForestMode            {1,0,0,0}
                    DomainMode            {1,1,1,0}
                    ReplicationSourceDC   {0,0,0,1}
                    SiteName              {0,1,1,1}
                    ParentDomainName      {0,1,1,0}
                    DomainName            {1,0,0,1}
                    DomainNetBIOSName     {1,0,0,0}
                    NewDomainName         {0,1,1,0}
                    NewDomainNetBIOSName  {0,1,1,0}

                })[$This.Mode]

                $This.Add($Name,$IsEnabled)
            }
        }
        [Object] ProfileSlotItem([UInt32]$Index,[String]$Name,[UInt32]$IsEnabled)
        {
            Return [ProfileSlotItem]::New($Index,$Name,$IsEnabled)
        }
        [UInt32] Index([String]$Name)
        {
            Return [UInt32][ProfileSlotType]::$Name
        }
        Add([String]$Name,[UInt32]$IsEnabled)
        {
            $This.Output += $This.ProfileSlotItem($This.Output.Count,$Name,$IsEnabled)
        }
        [String] ToString()
        {
            Return "({0}) <FEDCPromo.ProfileSlotList>" -f $This.Output.Count
        }
    }
```

Enum [ProfileRoleType]

These are the enumeration types for the possible roles that the command MAY or MAY NOT ask for, when initializing the command for domain controller promotion.

```
# (4/10) [Profile.Role.Type]
Enum ProfileRoleType
{
    InstallDns
    CreateDnsDelegation
    CriticalReplicationOnly
    NoGlobalCatalog
}
```

Class [ProfileRoleItem]

This allows the above enum types to be controllable in the same exact way as the “Slot” list items above.

```
# (5/10) [Profile.Role.Item]
Class ProfileRoleItem
{
    [UInt32]     $Index
    [String]      $Name
    [UInt32] $IsEnabled
    [UInt32] $IsChecked
    ProfileRoleItem([UInt32]$Index,[String]$Name,[Bool]$IsEnabled,[Bool]$IsChecked)
    {
        $This.Index     = $Index
        $This.Name      = $Name
        $This.IsEnabled = $IsEnabled
        $This.IsChecked = $IsChecked
    }
    Toggle()
    {
        $This.IsChecked = !$This.IsChecked
    }
    [String] ToString()
    {
        Return "<FEDCPromo.ProfileRoleItem>"
    }
}
```

Class [ProfileRoleList]

This allows each of those above roles to be set to their default values, as well as controlling them and whatever state they may be in.

```
# (6/10) [Profile.Role.List]
Class ProfileRoleList
{
    [UInt32]   $Mode
    [String]   $Name
    [Object] $Output
    ProfileRoleList([UInt32]$Mode)
```

```
        {
            $This.Mode = $Mode
            $This.Name = "Role"
            $This.Stage()
        }
        Clear()
        {
            $This.Output = @( )
        }
        Stage()
        {
            $This.Clear()

            ForEach ($Name in [System.Enum]::GetNames([ProfileRoleType]))
            {
                $Index = $This.Index($Name)
                $X     = Switch ($Name)
                {
                    InstallDNS              {(1,1,1,1),(1,1,1,1)}
                    CreateDNSDelegation     {(1,1,1,1),(0,0,1,0)}
                    NoGlobalCatalog         {(0,1,1,1),(0,0,0,0)}
                    CriticalReplicationOnly {(0,0,0,1),(0,0,0,0)}
                }

                $IsEnabled = $X[0][$This.Mode]
                $IsChecked = $X[1][$This.Mode]

                $This.Add($Index,$Name,$IsEnabled,$IsChecked)
            }
        }
        [Object] ProfileRoleItem([UInt32]$Index,[String]$Name,[UInt32]$IsEnabled,[UInt32]$IsChecked)
        {
            Return [ProfileRoleItem]::New($Index,$Name,$IsEnabled,$IsChecked)
        }
        [UInt32] Index([String]$Name)
        {
            Return [UInt32][ProfileRoleType]::$Name
        }
        Add([UInt32]$Index,[String]$Name,[UInt32]$IsEnabled,[UInt32]$IsChecked)
        {
            $This.Output += $This.ProfileRoleItem($Index,$Name,$IsEnabled,$IsChecked)
        }
        [String] ToString()
        {
            Return "({0}) <FEDCPromo.ProfileRoleList>" -f $This.Output.Count
        }
    }
```

_____/
_____/ Class [ProfileRoleList]
Enum [ProfilePasswordType] /_____\
/_____

This part here is meant to provide some control over the SafeModeAdministratorPassword.
If we establish some rules or criteria for the password, as well as some confirmation criteria, then we can make it a lot more difficult for someone to randomly spin up a domain controller by entering a password that was only asked for (1) time.

```
    # (7/10) [Profile.Password.Type]
    Enum ProfilePasswordType
    {
        Password
        Confirm
    }
```

_____/
_____/ Enum [ProfilePasswordType]

Class [ProfilePasswordItem]

This is effectively the same idea as the “slot” and “role” types, items, and lists above.
Though, this has a slightly different approach.

```
# (8/10) [Profile.Password.Item]
Class ProfilePasswordItem
{
    [UInt32]  $Index
    [String]   $Name
    [Object]  $Value
    [UInt32]  $Check
    [String] $Reason
    ProfilePasswordItem([UInt32]$Index,[String]$Name)
    {
        $This.Index = $Index
        $This.Name  = $Name
    }
    Validate([String]$Reason)
    {
        $This.Check  = [UInt32]($Reason -eq "[+] Passed")
        $This.Reason = $Reason
    }
    [String] ToString()
    {
        Return "<FEDCPromo.ProfilePasswordItem>"
    }
}
```

Class [ProfilePasswordList]

Same as the above lists.

```
# (9/10) [Profile.Password.List]
Class ProfilePasswordList
{
    [UInt32]   $Mode
    [String]   $Name
    [Object] $Output
    ProfilePasswordList([UInt32]$Mode)
    {
        $This.Mode = $Mode
        $This.Name = "Password"
        $This.Stage()
    }
    Clear()
    {
        $This.Output = @( )
    }
    Stage()
    {
        $This.Clear()

        ForEach ($Type in [System.Enum]::GetNames([ProfilePasswordType]))
        {
            $Index = $This.Index($Type)
            $This.Add($Index,$Type)
        }
    }
    [Object] ProfilePasswordItem([UInt32]$Index,[String]$Name)
    {
        Return [ProfilePasswordItem]::New($Index,$Name)
    }
    [UInt32] Index([String]$Name)
    {
```

```
            Return [UInt32][ProfilePasswordType]::$Name
        }
        Add([UInt32]$Index,[String]$Name)
        {
            $This.Output += $This.ProfilePasswordItem($Index,$Name)
        }
        [String] ToString()
        {
            Return "({0}) <FEDCPromo.ProfilePasswordList>" -f $This.Output.Count
        }
    }
```

Class [ProfilePasswordList]

Class [ProfileController]

So, whenever the command type is changed, this object right here is going to repopulate all of the available information, so that the Xaml object can be populated with those properties and values.

```
    # (10/10) [Profile.Controller]
    Class ProfileController
    {
        [UInt32]       $Index
        [String]        $Name
        [String]        $Type
        [String] $Description
        [Object]        $Slot
        [Object]        $Role
        [Object]        $DSRM
        ProfileController([Object]$Command)
        {
            If ($Command.Index -notin 0,1,2,3)
            {
                Throw "Invalid Entry"
            }

            $This.Index       = $Command.Index
            $This.Name        = $Command.Name
            $This.Type        = $Command.Type
            $This.Description = $Command.Description
            $This.Slot        = $This.New("Slot")
            $This.Role        = $This.New("Role")
            $This.Dsrm        = $This.New("DSRM")

            $This.SlotDefault()
        }
        SlotDefault()
        {
            ForEach ($Item in $This.Slot.Output)
            {
                $Item.Value = Switch ($Item.Name)
                {
                    ForestMode           { 0 }
                    DomainMode           { 0 }
                    ReplicationSourceDC  { 0 }
                    SiteName             { 0 }
                    ParentDomainName     { "<Enter Domain Name> or <Credential>"  }
                    DomainName           { "<Enter Domain Name> or <Credential>"  }
                    DomainNetBIOSName    { "<Enter NetBIOS Name> or <Credential>" }
                    NewDomainName        { "<Enter New Domain Name>"              }
                    NewDomainNetBIOSName { "<Enter New NetBIOS Name>"             }
                }
            }
        }
        [Object] New([String]$Name)
        {
            $Item = Switch ($Name)
            {
                Slot {     [ProfileSlotList]::New($This.Index) }
```

```
                Role {     [ProfileRoleList]::New($This.Index) }
                Dsrm { [ProfilePasswordList]::New($This.Index) }
            }

            Return $Item
        }
        [Object] Get([String]$Name)
        {
            $Item = Switch ($Name)
            {
                Slot { $This.Slot }
                Role { $This.Role }
                Dsrm { $This.Dsrm }
            }

            Return $Item
        }
        [Object] List([String]$Name)
        {
            Return $This.Get($Name).Output
        }
        [Object] Output()
        {
            $Out = @{ }
            ForEach ($Item in $This.List("Slot") | ? IsEnabled)
            {
                $Out.Add($Item.Name,$Item.Value)
            }
            ForEach ($Item in $This.List("Role") | ? IsEnabled)
            {
                $Out.Add($Item.Name,$Item.Value)
            }

            $Out.Add("SafeModeAdministratorPassword",$Null)
            $Out.Add("Credential",$Null)

            Return $Out
        }
        [String] ToString()
        {
            Return "<FEDCPromo.ProfileController[{0}]>" -f $This.Type
        }
    }
```

_____/ Class [ProfileController]
_____/ Enum [CommandType] /_____\

These are the command types, and everything about this utility revolves around these.

```
    # (1/3) [Command.Type]
    Enum CommandType
    {
        Forest
        Tree
        Child
        Clone
    }
```

_____/ Enum [CommandType]

Class [DomainTypeItem]

This changes the property DomainType based on the selected command type.

```
# (1/2) [Domain.Type.Item]
Class DomainTypeItem
{
    [UInt32] $Index
    [String]  $Name
    [String] $Value
    DomainTypeItem([String]$Name)
    {
        $This.Index = [UInt32][CommandType]::$Name
        $This.Name  = $Name
        $This.Value = @("-",$Name)[$Name -in "Tree","Child"]
    }
    [String] ToString()
    {
        Return $This.Name
    }
}
```

Class [DomainTypeList]

This is meant to categorize the domain types, though to be clear, this property isn't needed for Forest and Clone modes. It is specifically meant for the Tree and Child domain types.

```
# (2/2) [Domain.Type.List]
Class DomainTypeList
{
    [String]     $Name
    [UInt32] $Selected
    [Object]   $Output
    DomainTypeList()
    {
        $This.Name   = "DomainType"
        $This.Output = @( )

        ForEach ($Name in [System.Enum]::GetNames([CommandType]))
        {
            $This.Add($Name)
        }
    }
    [Object] DomainTypeItem([String]$Name)
    {
        Return [DomainTypeItem]::New($Name)
    }
    Add([String]$Name)
    {
        $This.Output += $This.DomainTypeItem($Name)
    }
    [Object] Current()
    {
        Return $This.Output[$This.Selected]
    }
    [String] ToString()
    {
        Return "<FEDCPromo.DomainTypeList>"
    }
}
```

Class [DomainTypeList]

Class [CommandTypeItem] /----------\
/------------------------

This is to enumerate the command types, as the domain types are a subset of the command type.

```
    # (2/3) [Command.Type.Item]
    Class CommandTypeItem
    {
        [UInt32]       $Index
        [String]        $Type
        [String]        $Name
        [String] $Description
        CommandTypeItem([UInt32]$Index,[String]$Type,[String]$Name,[String]$Description)
        {
            $This.Index       = $Index
            $This.Type        = $Type
            $This.Name        = $Name
            $This.Description = $Description
        }
        [String] ToString()
        {
            Return "<FEDCPromo.CommandTypeItem>"
        }
    }
```

\----------/
\----------/ Class [CommandTypeItem]
Class [CommandTypeList] /----------\
/------------------------

When the selected command is changed, then the name of the command and its' description will be changed as well.

```
    # (2/3) [Command.Type.List]
    Class CommandTypeList
    {
        [String]     $Name
        [UInt32] $Selected
        [Object]   $Output
        CommandTypeList()
        {
            $This.Name = "Command"
            $This.Stage()
        }
        Clear()
        {
            $This.Output = @( )
        }
        Stage()
        {
            $This.Clear()

            ForEach ($Type in [System.Enum]::GetNames([CommandType]))
            {
                $X     = Switch ($Type)
                {
                    Forest {           "Install-AddsForest" ,              "Creates a new Active Directory
forest" }
                    Tree   {           "Install-AddsDomain" ,         "Creates a new Active Directory tree
domain" }
                    Child  {           "Install-AddsDomain" ,        "Creates a new Active Directory child
domain" }
                    Clone  { "Install-AddsDomainController" , "Adds a new domain controller to an existing
domain" }
                }

                $This.Add($This.Index($Type),$Type,$X[0],$X[1])
            }
        }
        [Object] CommandTypeItem([UInt32]$Index,[String]$Type,[String]$Name,[String]$Description)
```

```
        {
            Return [CommandTypeItem]::New($Index,$Type,$Name,$Description)
        }
        [UInt32] Index([String]$Type)
        {
            Return [UInt32][CommandType]::$Type
        }
        Add([UInt32]$Index,[String]$Type,[String]$Name,[String]$Description)
        {
            $This.Output += $This.CommandTypeItem($Index,$Type,$Name,$Description)
        }
        [Object] Current()
        {
            Return $This.Output[$This.Selected]
        }
        [String] ToString()
        {
            Return "<FEDCPromo.CommandTypeList>"
        }
    }
```

Class [CommandController] / Class [CommandTypeList]

This effectively combines a number of the above classes into a subcontroller, allowing the Xaml object to have a range of input options and then providing some really granular and fine grained control over what items to render as COLLAPSED, or VISIBLE, as well as ENABLED and set to their default values.

```
    # (3/3) [Command.Controller]
    Class CommandController
    {
        [String]       $Name
        [UInt32]       $Slot = 0
        [Object]    $Command
        [Object] $DomainType
        [Object] $ForestMode
        [Object] $DomainMode
        [Object]    $Profile
        CommandController()
        {
            $This.Name    = "CommandController"
            $This.Stage()
            $This.SetProfile($This.Slot)
        }
        Clear()
        {
            $This.Command    = $Null
            $This.DomainType = $Null
            $This.ForestMode = $Null
            $This.DomainMode = $Null
            $This.Profile    = $Null
        }
        Stage()
        {
            $This.Clear()
            $This.Command    = $This.New("Command")
            $This.DomainType = $This.New("DomainType")
            $This.ForestMode = $This.New("ForestMode")
            $This.DomainMode = $This.New("DomainMode")
        }
        [Object] New([String]$Name)
        {
            $Item = Switch ($Name)
            {
                Command     {   [CommandTypeList]::New()             }
                DomainType  {    [DomainTypeList]::New()             }
                ForestMode  { [WindowsServerList]::New("ForestMode") }
                DomainMode  { [WindowsServerList]::New("DomainMode") }
```

```
            }

            Return $Item
        }
        [Object] Get([String]$Name)
        {
            $Item = Switch ($Name)
            {
                Command    { $This.Command    }
                DomainType { $This.DomainType }
                ForestMode { $This.ForestMode }
                DomainMode { $This.DomainMode }
            }

            Return $Item
        }
        [Object] List([String]$Name)
        {
            Return $This.Get($Name).Output
        }
        [Object] ProfileController([Object]$Object)
        {
            Return [ProfileController]::New($Object)
        }
        [Object] Current([String]$Name)
        {
            Return $This.Get($Name).Current()
        }
        SetProfile([UInt32]$Index)
        {
            $This.Slot               = $Index
            $This.Command.Selected   = $Index
            $This.DomainType.Selected = $Index
            $This.Profile            = $This.ProfileController($This.Current("Command"))
        }
        [String] ToString()
        {
            Return "<FEDCPromo.CommandController>"
        }
    }
```

_____/ Class [CommandController] _____/ Class [ConnectionItem] /_____\

This object right here is meant to collect information from an Active Directory connection, and this is only going to be used when a successful login to Active Directory is established.

```
    # (1/1) [Connection.Item]
    Class ConnectionItem
    {
        [String]        $IPAddress
        [String]          $DNSName
        [String]           $Domain
        [String]          $NetBIOS
        [PSCredential] $Credential
        Hidden [String]      $Site
        [String[]]       $Sitename
        [String[]]  $ReplicationDC
        ConnectionItem([Object]$Login)
        {
            $This.IPAddress            = $Login.IPAddress
            $This.DNSName              = $Login.DNSName
            $This.Domain               = $Login.Domain
            $This.NetBIOS              = $Login.NetBIOS
            $This.Credential           = $Login.Credential
            $This.Site                 = $Login.GetSitename()
            $Login.Directory           = $Login.Directory.Replace("CN=Partitions,","")
            $Login.Searcher.SearchRoot = $Login.Directory
```

```
            $Login.Result              = $Login.Searcher.FindAll()
            $This.Sitename             = @( )
            $This.Sitename            += $This.Site
            ForEach ($Item in $Login.Result | ? Path -Match "NTDS Site Settings")
            {
                $Item.Path.Split(",")[1].Replace("CN="","") | ? { $_ -ne $This.Site } | % { $This.Sitename += $_
}
            }
        }
        AddReplicationDCs([Object[]]$DCs)
        {
            $This.ReplicationDC        = $DCs.Hostname | Select-Object -Unique
        }
        [String] ToString()
        {
            Return "<FEDCPromo.Connection>"
        }
    }
```

_____/
_____/ Class [ConnectionItem]
Class [ValidationItem] /_____\
/_____

So, a portion of this utility has to check as to whether or not the input criteria for the text-based domain name stuff passes or fails, and this allows some of the restrictions in the possible number of testing criteria to be indexed and categorized.

```
    # (1/2) [Validation.Item]
    Class ValidationItem
    {
        [UInt32] $Index
        [String] $Type
        [String] $Value
        ValidationItem([UInt32]$Index,[String]$Type,[String]$Value)
        {
            $This.Index = $Index
            $This.Type  = $Type
            $This.Value = $Value
        }
        [String] ToString()
        {
            Return $This.Value
        }
    }
```

_____/
_____/ Class [ValidationItem]
Class [ValidationController] /_____\
/_____

This creates an object for validation, and having it in it's own controller allows the main controller to be somewhat less (complicated/complex).

```
    # (2/2) [Validation.Controller]
    Class ValidationController
    {
        [String]   $Name
        [Object] $Output
        ValidationController()
        {
            $This.Name   = "Validation"
            $This.Stage()
        }
        Clear()
        {
            $This.Output = @( )
```

```
        }
        Stage()
        {
            $This.Clear()
            ForEach ($Name in "Reserved","Legacy","SecurityDescriptor")
            {
                $This.Load($Name)
            }
        }
        Load([String]$Name)
        {
            ForEach ($Item in $This.Item($Name))
            {
                $This.Add($Name,$Item)
            }
        }
        [Object] ValidationItem([UInt32]$Index,[String]$Type,[String]$Value)
        {
            Return [ValidationItem]::New($Index,$Type,$Value)
        }
        Add([String]$Type,[String]$Value)
        {
            $This.Output += $This.ValidationItem($This.Output.Count,$Type,$Value)
        }
        [String[]] Item([String]$Name)
        {
            $Item = Switch ($Name)
            {
                Reserved
                {
                    "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"
                }
                Legacy
                {
                    "-GATEWAY;-GW;-TAC"
                }
                SecurityDescriptor
                {
                    "AN;AO;AU;BA;BG;BO;BU;CA;CD;CG;CO;DA;DC;DD;DG;DU;EA;ED;HI;IU;LA;LG;L"+
                    "S;LW;ME;MU;NO;NS;NU;PA;PO;PS;PU;RC;RD;RE;RO;RS;RU;SA;SI;SO;SU;SY;WD"
                }
            }

            Return $Item -Split ";"
        }
        [String] Password()
        {
            Return "(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[:punct:]).{10}"
        }
        [String] ToString()
        {
            Return "({0}) <FEDCPromo.ValidationController>" -f $This.Output.Count
        }
    }
```

_____/ Class [ValidationController]

Class [ExecutionController] /----------\
/----------

When the process successfully completes and whatnot, this object right here will collect the number of features that need to be installed, as well as the final promotion criteria. This also somewhat handles the ability for the process to use an InputObject, or even resume from a restart, if need be.

```
# (1/1) [Execution.Controller]
Class ExecutionController
{
    [String]     $Name
    [Object]  $Summary
    [Object]  $Feature
    [Object]   $Result
    [Object]   $Output
    ExecutionController()
    {
        $This.Name    = "Execution"
        $This.Clear("Summary")
        $This.Clear("Feature")
        $This.Clear("Result")
        $This.Clear("Output")
    }
    Clear([String]$name)
    {
        Switch ($Name)
        {
            Summary { $This.Summary = @( ) }
            Feature { $This.Feature = @( ) }
            Result  { $This.Result  = @( ) }
            Output  { $This.Output  = @{ } }
        }
    }
    [String] ToString()
    {
        Return "<FEDCPromo.ExecutionController>"
    }
}
```

\----------/ Class [ExecutionController]
Class [FEDCPromoController] /----------\
/----------

This class effectively combines all of the above classes, into a neat, tidy, scalable, controllable interface.

This thing... is pretty complicated.
I wanted it to contain a number of functions that are in the module, such as:
Get-FEModule, Get-FESystem, Get-FENetwork, and New-FEConsole.

Each of those functions is available in the module, and having all of this stuff bundled up together into the same package sort of makes the process usable and scalable.

```
# (1/1) [FEDCPromo.Controller]
Class FEDCPromoController
{
    [Object]    $Console
    [UInt32]       $Mode
    [UInt32]    $Staging
    [UInt32]       $Test
    [Object]       $Xaml
    [Object]     $Module
    [Object]     $System
    [Object]    $Network
    [String]    $Caption
    [UInt32]     $Server
    [Object]    $Feature
    [Object]    $Control
    [Object] $Connection
```

```
        [Object] $Credential
        [Object] $Validation
        [Object]  $Execution
        FEDCPromoController()
        {
            $This.Mode     = 0
            $This.Main()
        }
        FEDCPromoController([UInt32]$Mode)
        {
            $This.Mode     = $Mode
            If ($This.Mode -ge 2)
            {
                $This.Test = 1
            }
            $This.Main()
        }
        Main()
        {
            # Initialize console
            $This.StartConsole()

            # Load features
            $This.Feature  = $This.New("FEFeatureList")

            # Validate Adds installation
            $This.ValidateAdds()

            # Import Adds Module
            $This.ImportAdds()

            # Primary components
            $This.Module   = $This.New("FEModule")
            $This.System   = $This.New("FESystem")
            $This.Network  = $This.New("FENetwork")

            # Validate connectivity, and whether DHCP is set
            $Check         = $This.System.Network.Output | ? Status
            Switch ($Check.Count)
            {
                0
                {
                    Write-Theme "Error [!] No network detected" 1
                    Break
                }
                1
                {
                    If ($Check[0].DhcpServer -notmatch "(\d+\.){3}\d+")
                    {
                        $This.Update(0,"Warning [!] Static IP Address not set")
                    }
                }
                Default
                {
                    If ($Check.DhcpServer -notmatch "(\d+\.){3}\d+")
                    {
                        $This.Update(0,"Warning [!] Static IP Address not set")
                    }
                }
            }

            # Check if system is a virtual machine
            If ($This.System.ComputerSystem.Model -match "Virtual")
            {
                $This.Update(0,"Detected [!] Current system is a virtual machine")
                $This.Module.Write(1,$This.Console.Last().Status)

                ForEach ($Item in $This.Feature.Output | ? Type -eq Veridian)
                {
                    $Item.Enable  = 0
                    $Item.Install = 0
                }
```

```
        }

        # Server information
        $This.Caption  = $This.System.OperatingSystem.Caption
        $This.Server   = Switch -Regex ($This.Caption)
        {
            "(2000)"        { 0 } "(2003)"        { 1 } "(2008 (?!R2))" { 2 }
            "(2008 R2)"     { 3 } "(2012 (?!R2))" { 4 } "(2012 R2)"     { 5 }
            "(2016)"        { 6 } "(2019)"        { 7 } "(2022)"        { 8 }
        }

        If ($This.Server -le 3)
        {
            Write-Error "<This operating system may be too old to support this function>"
        }

        # Get Command/Profile controller
        $This.Control       = $This.New("CommandController")

        # Stage command (Forest/Domain) modes
        $This.Control.Get("ForestMode").SetMin($This.Server)
        $This.Control.Get("DomainMode").SetMin($This.Server)

        # Get validation controller
        $This.Validation    = $This.New("ValidationController")

        # Load execution controller
        $This.Execution     = $This.New("ExecutionController")

        # Load Xaml
        $This.Xaml          = $This.New("FEDCPromoXaml")

        # Continue with [Xaml]
        $This.StageXaml()
    }
    StartConsole()
    {
        # Instantiates and initializes the console
        $This.Console = New-FEConsole
        $This.Console.Initialize()
        $This.Status()
    }
    [Void] Status()
    {
        # If enabled, shows the last item added to the console
        If ($This.Mode -gt 0)
        {
            [Console]::WriteLine($This.Console.Last())
        }
    }
    [Void] Update([Int32]$State,[String]$Status)
    {
        # Updates the console
        $This.Console.Update($State,$Status)
        $This.Status()
    }
    [String] Adds()
    {
        Return "Module: [AddsDeployment]"
    }
    [String] ProgramData()
    {
        Return [Environment]::GetEnvironmentVariable("ProgramData")
    }
    [UInt32] TestPath([String]$Path)
    {
        Return [System.IO.Directory]::Exists($Path)
    }
    [String] OutputFolder()
    {
        $List = $This.ProgramData(),
                "Secure Digits Plus LLC",
```

```
                    "FEDCPromo",
                    $This.Console.Start.Time.ToString("yyyyMMdd")

            $Path = $List -join "\"

            If (!$This.TestPath($Path))
            {
                $Path = $List[0]
                ForEach ($Item in $List[1..3])
                {
                    $Path += "\$Item"
                    If (!$This.TestPath($Path))
                    {
                        [System.IO.Directory]::CreateDirectory($Path) | Out-Null
                    }
                }
            }

            Return $Path
        }
        ExportFile([String]$Type,[Object]$Object)
        {
            $FileName = Switch ($Type)
            {
                Feature     {     "Feature.json" }
                Control     {     "Control.csv"  }
                InputObject { "InputObject.csv"  }
                Credential  {  "Credential.txt"  }
                Dsrm        {        "Dsrm.txt"  }
                Script      {      "Script.ps1"  }
            }

            $Path = "{0}\{1}" -f $This.OutputFolder(), $Filename

            Switch -Regex ($Type)
            {
                "(^Feature$|^Control$|^InputObject$)"
                {
                    $Value = ConvertTo-Json $Object
                    [System.IO.File]::WriteAllLines($Path,$Value)
                }
                "(^Credential$|^Dsrm$)"
                {
                    Export-CliXml -Path $Path -InputObject $Object -Force
                }
                Script
                {
                    [System.IO.File]::WriteAllLines($Path,$Object)
                }
            }

            Switch ([UInt32][System.IO.File]::Exists($Path))
            {
                0 { $This.Update(-1,"Failed [!] Type: [$Type], File: [$Path]") }
                1 { $This.Update( 1, "Saved [+] Type: [$Type], File: [$Path]") }
            }
        }
        ValidateAdds()
        {
            # (Validates/installs) the [AddsDeployment] module
            $AddsStr = $This.Adds()
            $Adds    = $This.Feature.Output | ? Name -eq AD-Domain-Services
            Switch ($Adds.State)
            {
                0
                {
                    # [AddsDeployment] not detected, prompt for choice
                    (Get-Host).UI.PromptForChoice("$AddsStr was not detected.","Install
$AddsStr",@("Yes","No"),0)
                    {
                        0
                        {
```

```
                        # User did not proceed with [AddsDeployment] installation
                        $This.Update(-1,"Exception [!] Install -> $AddsStr")
                        Throw $This.Console.Last().Status
                    }
                    1
                    {
                        # User did proceed with [AddsDeployment] installation
                        $This.Update(0,"Process [~] Install -> $AddsStr")
                        Write-Theme $This.Console.Last().Status
                        $This.InstallAdds()

                        # Checks and handles whether the installation completed
                        If (!(Get-Module AddsDeployment))
                        {
                            $This.Update(-1,"Exception [!] Install -> $AddsStr")
                            Throw $This.Console.Last().Status
                        }
                        Else
                        {
                            $This.Update(1,"Success [+] Install -> $AddsStr")
                            $Adds.State = 1
                        }
                    }
                }
            }
            1
            {
                # [AddsDeployment] detected
                $This.Update(1,"Valid [+] $AddsStr")
            }
        }
    }
    InstallAdds()
    {
        # Installs the [AddsDeployment] module
        Install-WindowsFeature Ad-Domain-Services -Confirm:$False
    }
    [Object] InstallWindowsFeature([String]$Name)
    {
        # Installs a specified feature
        Return Install-WindowsFeature -Name $Name -IncludeAllSubfeature -IncludeManagementTools
    }
    ImportAdds()
    {
        # Imports the [AddsDeployment] module
        $This.Update(0,"Importing [~] Module: [AddsDeployment]")
        Import-Module AddsDeployment
    }
    [Object] New([String]$Name)
    {
        # Returns an instantiation of the named (function/class)
        $Item = Switch ($Name)
        {
            FEModule             { Get-FEModule -Mode 1                }
            FESystem             { Get-FESystem -Mode 0 -Level 3       }
            FENetwork            { Get-FENetwork -Mode 7               }
            FEADLogin            { Get-FEADLogin                       }
            FEFeatureList        {         [FeatureController]::New()  }
            CommandController    {         [CommandController]::New()  }
            ValidationController {      [ValidationController]::New()  }
            ExecutionController  {       [ExecutionController]::New()  }
            FEDCPromoXaml        { [XamlWindow][FEDCPromoXaml]::Content }
            FEDCFoundXaml        { [XamlWindow][FEDCFoundXaml]::Content }
        }

        # Logs the instantiation of the named (function/class)
        Switch ([UInt32]!!$Item)
        {
            0 { $This.Update(-1,"Failed [!] [$Name]") }
            1 { $This.Update( 1,"Loaded [+] [$Name]") }
        }
```

```
        Return $Item
    }
    [Object] InstallDomainController([String]$Name,[Hashtable]$Splat)
    {
        # Installs/Tests the domain controller promotion
        $This.Update(0,"Attempting [~] [$Name]")
        $Item = Switch ($Name)
        {
            AddsForest               {                   Install-ADDSForest @Splat -Confirm:$False }
            AddsDomain               {                   Install-ADDSDomain @Splat -Confirm:$False }
            AddsDomainController     {          Install-ADDSDomainController @Splat -Confirm:$False }
            TestAddsForest           {           Test-ADDSForestInstallation @Splat                 }
            TestAddsDomain           {           Test-ADDSDomainInstallation @Splat                 }
            TestAddsDomainController { Test-ADDSDomainControllerInstallation @Splat                 }
        }

        Return $Item
    }
    [Object] GetConnection([Object]$Connect)
    {
        # Returns a connection object from a successful AD connection
        Return [ConnectionItem]::New($Connect)
    }
    [Object] Slot([String]$Name)
    {
        # Returns the specified slot item from profile controller
        Return $This.Control.Profile.Slot.Output | ? Name -eq $Name
    }
    [Object] Role([String]$Name)
    {
        # Returns the specified role item from profile controller
        Return $This.Control.Profile.Role.Output | ? Name -eq $Name
    }
    [Object] Dsrm([String]$Name)
    {
        # Returns the (password/confirm) item from profile controller
        Return $This.Control.Profile.Dsrm.Output | ? Name -eq $Name
    }
    [String] SystemRoot()
    {
        # Returns the system root path
        Return [Environment]::GetEnvironmentVariable("SystemRoot")
    }
    [String] ComputerName()
    {
        # Returns the computer name
        Return [Environment]::MachineName
    }
    [String] Icon([UInt32]$Type)
    {
        # Returns the (success/failure) graphic based on the type
        Return $This.Module._Control(@("failure.png","success.png")[$Type]).Fullname
    }
    [Object] Validate([String]$Type)
    {
        # Retrieves a list of specific item types from the validation controller
        Return $This.Validation.Output | ? Type -eq $Type | % Value
    }
    [Object] Reserved()
    {
        # Returns reserved items from the validation controller
        Return $This.Validate("Reserved")
    }
    [Object] Legacy()
    {
        # Returns legacy items from the validation controller
        Return $This.Validate("Legacy")
    }
    [Object] SecurityDescriptor()
    {
        # Returns security descriptor items from the validation controller
        Return $This.Validate("SecurityDescriptor")
```

```
        }
        [String] DefaultText([String]$Name)
        {
            # Returns the default string for the properties below
            $Item = Switch ($Name)
            {
                ParentDomainName     { "<Enter Domain Name> or <Credential>"  }
                DomainName           { "<Enter Domain Name> or <Credential>"  }
                DomainNetBIOSName    { "<Enter NetBIOS Name> or <Credential>" }
                NewDomainName        { "<Enter New Domain Name>"              }
                NewDomainNetBIOSName { "<Enter New NetBIOS Name>"             }
            }

            Return $Item
        }
        [String] DefaultPath([String]$Name)
        {
            # Returns the default path string of the properties below
            $Item = Switch ($Name)
            {
                DataBasePath { "{0}\NTDS"    }
                SysVolPath   { "{0}\SYSVOL"  }
                LogPath      { "{0}\NTDS"    }
            }

            Return $Item -f $This.SystemRoot()
        }
        Browse([String]$Name)
        {
            # Opens the folder browser dialog for the paths below
            If ($Name -notin "DatabasePath","SysvolPath","Logpath")
            {
                Throw "Invalid item"
            }

            $Item                    = New-Object System.Windows.Forms.FolderBrowserDialog
            $Item.ShowDialog()

            If (!$Item.SelectedPath)
            {
                $Item.SelectedPath   = $This.DefaultPath($Name)
            }

            $This.Xaml.IO.$Name.Text = $Item.SelectedPath
        }
        Reset([Object]$xSender,[Object]$Object)
        {
            # Resets the items within a given combobox or datagrid
            If ($This.Mode -gt 0)
            {
                $Line = "Xaml.IO.{0} [{1}]" -f $xSender.Name, $xSender.GetType().Name
                $This.Update(0,"Resetting [~] $Line")
            }

            $xSender.Items.Clear()
            ForEach ($Item in $Object)
            {
                $xSender.Items.Add($Item)
            }
        }
        XamlDefault([String]$Section)
        {
            # Sets various default settings for the Xaml controller
            $This.Update(0,"Setting [~] Defaults: [$Section]")
            Switch ($Section)
            {
                ComboBox
                {
                    $This.Xaml.IO.ForestMode.IsEnabled          = 0
                    $This.Xaml.IO.DomainMode.IsEnabled          = 0
                    $This.Xaml.IO.ReplicationSourceDc.IsEnabled = 0
                    $This.Xaml.IO.SiteName.IsEnabled            = 0
```

```
            }
            Credential
            {
                $This.Xaml.IO.Credential.IsEnabled                    = 0
                $This.Xaml.IO.SafeModeAdministratorPassword.IsEnabled = 0
                $This.Xaml.IO.Confirm.IsEnabled                       = 0
            }
            Name
            {
                $This.Xaml.IO.DomainName.IsEnabled                    = 0
                $This.Xaml.IO.NewDomainName.IsEnabled                 = 0
                $This.Xaml.IO.DomainNetBIOSName.IsEnabled             = 0
                $This.Xaml.IO.NewDomainNetBIOSName.IsEnabled          = 0
                $This.Xaml.IO.SiteName.IsEnabled                      = 0
            }
            Path
            {
                $This.Xaml.IO.DataBasePath.Text  = $This.DefaultPath("DatabasePath")
                $This.Xaml.IO.SysVolPath.Text    = $This.DefaultPath("SysvolPath")
                $This.Xaml.IO.LogPath.Text       = $This.DefaultPath("LogPath")
            }
            Button
            {
                $This.Xaml.IO.Start.IsEnabled                         = 0
                $This.Xaml.IO.Test.IsEnabled                          = 0
                $This.Xaml.IO.CredentialButton.IsEnabled              = 0
            }
        }
    }
    ToggleRole([String]$Name)
    {
        # Toggles the specified checkbox
        $Item               = $This.Role($Name)

        If ($Item.IsEnabled)
        {
            $This.Update(0,"Toggling [~] [$Name]")
            $Item.Toggle()
        }
    }
    ToggleStaging()
    {
        # Toggles whether event handlers are (engaged/suppressed)
        $This.Staging = !$This.Staging
    }
    SetProfile([UInt32]$Index)
    {
        # Whenever the profile is changed, this recycles all of the controls
        $This.Update(0,"Changed [+] Selection [$Index]")
        $This.Control.SetProfile($Index)

        # Enabled staging mode [prevents event handler spamming]
        $This.ToggleStaging()

        # DomainType
        $This.SetDomainType()

        # Credential
        $This.SetCredential()

        # Features
        $This.SetFeatures()

        # Roles
        $This.SetRoles()

        # Slots
        $This.SetSlots()

        # Connection
        $This.SetConnection()
```

```
            # Administrator password entry
            $This.SetAdminPassword()

            # Disable staging mode
            $This.ToggleStaging()
        }
        SetMode()
        {
            $This.Update(0,"Setting [~] (Forest/Domain) modes")
            $This.Slot("ForestMode").Value = $This.Control.Get("ForestMode").Selected
            $This.Slot("DomainMode").Value = $This.Control.Get("DomainMode").Selected
        }
        SetDomainType()
        {
            $This.Update(0,"Setting [~] DomainType")
            $This.Control.Get("DomainType").Selected = $This.Control.Slot
        }
        SetCredential()
        {
            $This.Update(0,"Setting [~] Credential button")
            $This.Xaml.IO.CredentialButton.IsEnabled = $This.Control.Slot -ne 0
            $This.Xaml.IO.CredentialBox.IsEnabled    = $This.Control.Slot -ne 0
        }
        SetFeatures()
        {
            $This.Update(0,"Settings [~] Features")
        }
        SetRoles()
        {
            $This.Update(0,"Setting [~] Roles")
            ForEach ($Item in $This.Control.Profile.List("Role"))
            {
                $Name             = $Item.Name
                $Enabled          = $Item.IsEnabled
                $Checked          = $Item.IsChecked
                $Mark             = @(" ","X")[$Enabled]

                $This.Update(0,"Setting [~] Role [$Mark], Name: [$Name], Enabled: [$Enabled], Checked:
[$Checked]")

                $Object           = $This.Xaml.IO.$Name
                $Object.IsEnabled = $Enabled
                $Object.IsChecked = $Checked
            }
        }
        SetSlots()
        {
            $This.Update(0,"Setting [~] Profile slot")
            ForEach ($Type in "ComboBox","TextBox")
            {
                ForEach ($Item in $This.Control.Profile.List("Slot") | ? Type -eq $Type)
                {
                    $Name               = $Item.Name
                    $Object             = $This.Xaml.IO.$Name
                    $Box                = $This.Xaml.IO."$Name`Box"
                    $Rank               = $Item.IsEnabled
                    $Mark               = @(" ","X")[$Rank]

                    $This.Update(0,"Setting [~] Profile [$Mark], Slot: [$Name], Type: [$Type]")

                    $Object.Visibility  = @("Collapsed","Visible")[$Rank]
                    $Object.IsEnabled   = $Rank
                    $Box.Visibility     = @("Collapsed","Visible")[$Rank]
                    $Box.IsEnabled      = $Rank

                    Switch ($Type)
                    {
                        ComboBox
                        {
                            $Item.Value = @(0,$This.Server)[$Item.Name -match "Mode"]
                        }
                        TextBox
```

```
                    {
                        $Item.Value = @("",$Item.Value)[$Item.IsEnabled]
                    }
                }

                If ($Type -eq "ComboBox")
                {
                    If ($Item.IsEnabled)
                    {
                        $Object.SelectedIndex = @($Item.Value,$This.Server)[$Name -match "Mode"]
                        $Object.IsEnabled     = 1
                    }
                    Else
                    {
                        $Object.SelectedIndex = 0
                    }
                }
                If ($Type -eq "TextBox")
                {
                    $Icon                               = "{0}Icon" -f $Name

                    If ($Item.IsEnabled)
                    {
                        $Object.Text                    = $Item.Value
                        $Object.IsEnabled               = 1

                        $This.Xaml.IO.$Icon.Visibility = "Visible"
                    }
                    Else
                    {
                        If ($Object.Text)
                        {
                            $Object.Text                = ""
                        }

                        $Icon                           = "{0}Icon" -f $Name
                        $This.Xaml.IO.$Icon.Visibility = "Collapsed"
                        $This.Xaml.IO.$Icon.Source     = $Null
                    }
                }
            }
        }
    }
    SetConnection()
    {
        $This.Update(0,"Setting [~] Connection")
        If ($This.Connection)
        {
            $This.Update(0,"Detected [+] Connction")
            Switch ($This.Control.Slot)
            {
                0
                {
                    $This.Connection = $Null
                }

                1
                {
                    $This.Slot("ParentDomainName" ).Value = $This.Connection.Domain
                    $This.Slot("DomainNetBIOSName").Value = $This.Connection.NetBIOS
                }

                2
                {
                    $This.Slot("ParentDomainName" ).Value = $This.Connection.Domain
                    $This.Slot("DomainNetBIOSName").Value = $This.Connection.NetBIOS
                }

                3
                {
                    $This.Slot("DomainName"       ).Value = $This.Connection.Domain
                    $This.Slot("DomainNetBIOSName").Value = $This.Connection.NetBIOS
```

```
                }
            }
        }

        # Connection Objects [Credential, Sitename, and ReplicationDCs]
        If ($This.Control.Slot -eq 0)
        {
            $This.Update(0,"Clearing [~] Credential")
            $This.Xaml.IO.Credential.Text                       = ""
            $This.Xaml.IO.CredentialButton.IsEnabled            = 0
            $This.Credential                                    = $Null
        }

        If ($This.Control.Slot -ne 0 -and $This.Connection)
        {
            $This.Update(0,"Selecting [~] Credential")
            $This.Xaml.IO.Credential.Text                       = $This.Connection.Credential.Username
            $This.Credential                                    = $This.Connection.Credential
            $This.Xaml.IO.CredentialButton.IsEnabled            = 1

            $Item = Switch ($This.Connection.Sitename.Count)
            {
                0 { "-" } Default { $This.Connection.Sitename }
            }

            $This.Reset($This.Xaml.IO.SiteName,$Item)

            $Item = Switch ($This.Connection.ReplicationSourceDC.Count)
            {
                0 { "<Any>" } Default { @($This.Connection.ReplicationDC;"<Any>") }
            }

            $This.Reset($This.Xaml.IO.ReplicationSourceDC,$Item)

            $This.Xaml.IO.Sitename.SelectedIndex                = 0
            $This.Xaml.IO.ReplicationSourceDC.SelectedIndex     = 0
        }
    }
    SetAdminPassword()
    {
        $This.Update(0,"Setting [~] DSRM password")
        $This.Xaml.IO.SafeModeAdministratorPassword.IsEnabled   = 1
        $This.Xaml.IO.Confirm.IsEnabled                         = 1
    }
    Login()
    {
        $This.Update(0,"Initializing [~] Ad login")
        $This.Connection         = $Null
        $Dcs                     = $This.Network.NBT.Output
                                   # $Ctrl.Network.Compartment.Output[0].Extension.Nbt.Output.Output | FT
                                   # $This.Network.Nbt.Output | ? {$_.Output.Id -match "(1B|1C)" }
                                   # $This.Network.Compartment
        Switch ([UInt32]!!$Dcs)
        {
            0
            {
                $Connect         = $This.Get("FEADLogin")
                $This.Connection = Switch ([UInt32]!!$Connect.Test.DistinguishedName)
                {
                    0
                    {
                        $This.Update(-1,"Failed [!] Ad login")
                        $Null
                    }
                    1
                    {
                        $This.Update(1,"Success [+] Ad login")
                        $This.GetConnection($Connect)
                    }
                }
            }
            1
```

```
            {
                $DC              = $This.Get("FDCFoundXaml")

                $This.Reset($DC.IO.DomainControllers,$DCs)
                $DC.IO.DomainControllers.Add_SelectionChanged(
                {
                    If ($DC.IO.DomainControllers.SelectedIndex -ne -1)
                    {
                        $DC.IO.Ok.IsEnabled = 1
                    }
                })

                $DC.IO.Ok.IsEnabled     = 0
                $DC.IO.Cancel.Add_Click(
                {
                    $DC.IO.DialogResult = 0
                })

                $DC.IO.Ok.Add_Click(
                {
                    $DC.IO.DialogResult = 1
                })

                $DC.Invoke()

                Switch ($DC.IO.DialogResult)
                {
                    0
                    {
                        $This.Update(-1,"Exception [!] Ad login: (user cancelled/dialog failed)")
                    }
                    1
                    {
                        $This.Update(-1,"Attempting [~] Ad login: testing supplied credentials")
                        $Connect = Get-FEADLogin -Target $DC.IO.DomainControllers.SelectedItem
                        Switch ([UInt32]!!$Connect.Test.DistinguishedName)
                        {
                            0
                            {
                                $This.Update(-1,"Failed [!] Ad login: credential error")
                                $This.Connection = $Null
                            }
                            1
                            {
                                $This.Update(1,"Success [+] Ad login: credential good")
                                $This.Connection = $This.GetConnection($Connect)
                                $This.Connection.AddReplicationDCs($DCs)
                            }
                        }
                    }
                }
            }
        }

        $This.SetProfile($This.Control.Slot)
    }
    Check([String]$Name)
    {
        $Item = $This.Slot($Name)
        $This.Check($Item)
    }
    Check([Object]$Item)
    {
        If (!$Item)
        {
            Throw "Null entry"
        }

        $Name = $Item.Name
        $Icon = "{0}Icon" -f $Name

        If (!$This.Staging)
```

```
        {
            If ($Item.IsEnabled)
            {
                $This.Update(0,"Checking [~] Field: [$Name]")
                $Item.Value                 = $This.Xaml.IO.$Name.Text

                Switch ($Name)
                {
                    {$_ -in "ParentDomainName","DomainName"}
                    {
                        $This.CheckItem("Domain",$Item)
                    }
                    {$_ -in "DomainNetBIOSName","NewDomainNetBIOSName"}
                    {
                        $This.CheckItem("NetBIOS",$Item)
                    }
                    {$_ -eq "NewDomainName" -and $This.Control.Slot -eq 1}
                    {
                        $This.CheckItem("Tree",$Item)
                    }
                    {$_ -eq "NewDomainName" -and $This.Control.Slot -eq 2}
                    {
                        $This.CheckItem("Child",$Item)
                    }
                }

                $This.Xaml.IO.$Name.Visibility = "Visible"
                $This.Xaml.IO.$Icon.Source     = $This.Icon($Item.Check)
                $This.Xaml.IO.$Icon.Tooltip    = $Item.Reason
                $This.Xaml.IO.$Icon.Visibility = "Visible"
            }

            If (!$Item.IsEnabled)
            {
                $This.Xaml.IO.$Name.Visibility = "Collapsed"
                $This.Xaml.IO.$Icon.Source     = $Null
                $This.Xaml.IO.$Icon.Tooltip    = $Null
                $This.Xaml.IO.$Icon.Visibility = "Collapsed"
            }

            $This.Total()
        }
    }
    CheckItem([String]$Type,[Object]$Item)
    {
        $X = $Null
        Switch ($Type)
        {
            Domain
            {
                If ($Item.Value.Length -lt 2 -or $Item.Value.Length -gt 63)
                {
                    $X = "[!] Length not between 2 and 63 characters"
                }
                ElseIf ($Item.Value -in $This.Reserved())
                {
                    $X = "[!] Entry is in reserved words list"
                }
                ElseIf ($Item.Value -in $This.Legacy())
                {
                    $X = "[!] Entry is in the legacy words list"
                }
                ElseIf ($Item.Value -notmatch "([\.\-0-9a-zA-Z])")
                {
                    $X = "[!] Invalid characters"
                }
                ElseIf ($Item.Value[0,-1] -match "(\W)")
                {
                    $X = "[!] First/Last Character cannot be a '.' or '-'"
                }
                ElseIf ($Item.Value.Split(".").Count -lt 2)
                {
```

```
                    $X = "[!] Single label domain names are disabled"
                }
                ElseIf ($Item.Value.Split('.')[-1] -notmatch "\w")
                {
                    $X = "[!] Top Level Domain must contain a non-numeric"
                }
                Else
                {
                    $X = "[+] Passed"
                }
            }
            NetBios
            {
                If ($Item.Value -eq $This.Connection.NetBIOS)
                {
                    $X = "[!] New NetBIOS ID cannot be the same as the parent domain NetBIOS"
                }
                ElseIf ($Item.Value.Length -lt 1 -or $Item.Value.Length -gt 15)
                {
                    $X = "[!] Length not between 1 and 15 characters"
                }
                ElseIf ($Item.Value -in $This.Reserved())
                {
                    $X = "[!] Entry is in reserved words list"
                }
                ElseIf ($Item.Value -in $This.Legacy())
                {
                    $X = "[!] Entry is in the legacy words list"
                }
                ElseIf ($Item.Value -notmatch "([\.\-0-9a-zA-Z])")
                {
                    $X = "[!] Invalid characters"
                }
                ElseIf ($Item.Value[0,-1] -match "(\W)")
                {
                    $X = "[!] First/Last Character cannot be a '.' or '-'"
                }
                ElseIf ($Item.Value -match "\.")
                {
                    $X = "[!] NetBIOS cannot contain a '.'"
                }
                ElseIf ($Item.Value -in $This.SecurityDescriptor())
                {
                    $X = "[!] Matches a security descriptor"
                }
                Else
                {
                    $X = "[+] Passed"
                }
            }
            Tree
            {
                If ($Item.Value -match [Regex]::Escape($This.Xaml.IO.ParentDomainName.Text))
                {
                    $X = "[!] Cannot be a (child/host) of the parent"
                }
                ElseIf ($Item.Value.Split(".").Count -lt 2)
                {
                    $X = "[!] Single label domain names are disabled"
                }
                ElseIf ($Item.Value.Split('.')[-1] -notmatch "\w")
                {
                    $X = "[!] Top Level Domain must contain a non-numeric"
                }
                ElseIf ($Item.Value.Length -lt 2 -or $Item.Value.Length -gt 63)
                {
                    $X = "[!] Length not between 2 and 63 characters"
                }
                ElseIf ($Item.Value -in $This.Reserved())
                {
                    $X = "[!] Entry is in reserved words list"
                }
```

```
                    ElseIf ($Item.Value -in $This.Legacy())
                    {
                        $X = "[!] Entry is in the legacy words list"
                    }
                    ElseIf ($Item.Value -notmatch "([\.\-0-9a-zA-Z])")
                    {
                        $X = "[!] Invalid characters"
                    }
                    ElseIf ($Item.Value[0,-1] -match "(\W)")
                    {
                        $X = "[!] First/Last Character cannot be a '.' or '-'"
                    }
                    Else
                    {
                        $X = "[+] Passed"
                    }
                }
                Child
                {
                    If ($Item.Value -notmatch ".$($This.Xaml.IO.ParentDomainName.Text)")
                    {
                        $X = "[!] Must be a (child/host) of the parent"
                    }
                    ElseIf ($Item.Value.Length -lt 2 -or $Item.Value.Length -gt 63)
                    {
                        $X = "[!] Length not between 2 and 63 characters"
                    }
                    ElseIf ($Item.Value -in $This.Reserved())
                    {
                        $X = "[!] Entry is in reserved words list"
                    }
                    ElseIf ($Item.Value -in $This.Legacy())
                    {
                        $X = "[!] Entry is in the legacy words list"
                    }
                    ElseIf ($Item.Value -notmatch "([\.\-0-9a-zA-Z])")
                    {
                        $X = "[!] Invalid characters"
                    }
                    ElseIf ($Item.Value[0,-1] -match "(\W)")
                    {
                        $X = "[!] First/Last Character cannot be a '.' or '-'"
                    }
                    Else
                    {
                        $X = "[+] Passed"
                    }
                }
            }

            $Item.Validate($X)
            Switch ([UInt32]!!$Item.Check)
            {
                0 { $This.Update(-1, "Failed $($Item.Reason)") }
                1 { $This.Update( 1,"Success $($Item.Reason)") }
            }
        }
        CheckDsrmPassword()
        {
            $Pattern  = $This.Validation.Password()
            $Password = $This.Dsrm("Password")

            Switch -Regex ($Password.Value)
            {
                Default
                {
                    $Password.Validate("[!] 10 chars, and at least: (1) Uppercase, (1) Lowercase, (1) Special,
(1) Number")
                }
                $Pattern
                {
                    $Password.Validate("[+] Passed")
```

```
            }
        }
    }
    CheckDSRMConfirm()
    {
        $This.CheckDSRMPassword()
        $Password = $This.DSRM("Password")
        $Confirm  = $This.DSRM("Confirm")

        If ($Password.Check -eq 0)
        {
            $Confirm.Validate("[!] Password not valid")
        }
        ElseIf ($Password.Check -eq 1 -and $Confirm.Value -ne $Password.Value)
        {
            $Confirm.Validate("[!] Confirmation error")
        }
        ElseIf ($Password.Check -eq 1 -and $Confirm.Value -eq $Password.Value)
        {
            $Confirm.Validate("[+] Passed")
        }
    }
    Total()
    {
        $This.Execution.Clear("Summary")

        ForEach ($Item in $This.Control.Profile.List("Slot") | ? IsEnabled | ? Property -eq Text)
        {
            $This.Execution.Summary += $Item
        }

        ForEach ($Item in $This.Control.Profile.List("DSRM"))
        {
            $This.Execution.Summary += $Item
        }

        $This.Reset($This.Xaml.IO.Summary,$This.Execution.Summary)

        $This.Xaml.IO.Start.IsEnabled = 0 -notin $This.Execution.Summary.Check
        $This.Xaml.IO.Test.IsEnabled  = 0 -notin $This.Execution.Summary.Check
    }
    Complete()
    {
        $Item               = $This.Slot("ForestMode")
        If ($Item.IsEnabled)
        {
            $Index          = $This.Xaml.IO.ForestMode.SelectedIndex
            $Item.Value     = @($Index,"WinThreshold")[$Index -ge 6]
        }

        $Item               = $This.Slot("DomainMode")
        If ($Item.IsEnabled)
        {
            $Index          = $This.Xaml.IO.DomainMode.SelectedIndex
            $Item.Value     = @($Index,"WinThreshold")[$Index -ge 6]
        }

        $Item               = $This.Slot("ReplicationSourceDC")
        If ($Item.IsEnabled)
        {
            $Item.Value     = $This.Xaml.IO.ReplicationSourceDC.SelectedItem
        }

        $Item               = $This.Slot("SiteName")
        If ($Item.IsEnabled)
        {
            $Item.Value     = $This.Xaml.IO.Sitename.SelectedItem
        }

        ForEach ($Item in $This.Control.Profile.List("Role"))
        {
            $Name           = $Item.Name
```

```
                $Item.IsChecked = $Item.IsEnabled -and $This.Xaml.IO.$Name.IsChecked
            }

            If ($This.Control.Slot -eq 2)
            {
                $Item           = $This.Slot("NewDomainName")
                $Item.Value     = $Item.Value.Replace($This.Connection.Domain,"").TrimEnd(".")
            }

            $Item               = $This.Dsrm("Password")
            $Item.Value         = $This.Xaml.IO.SafeModeAdministratorPassword.SecurePassword
            $Item               = $This.Dsrm("Confirm")
            $Item.Value         = $This.Xaml.IO.Confirm.SecurePassword
        }
        DumpConsole()
        {
            $This.Console.Finalize()
            $List = $This.ProgramData(),
                    "Secure Digits Plus LLC",
                    "Logs"

            $Path = $List -join "\"

            If (!$This.TestPath($Path))
            {
                $Path = $List[0]
                ForEach ($Item in $List[1..2])
                {
                    $Path += "\$Item"
                    If (!$This.TestPath($Path))
                    {
                        [System.IO.Directory]::CreateDirectory($Path) | Out-Null
                    }
                }
            }

            $FileName = "{0}\{1}.log" -f $Path, $This.Console.End.Time.ToString("yyyyMMdd")
            $Value    = $This.Console.Output | % ToString

            [System.IO.File]::WriteAllLines($FileName,$Value)
        }
        [Object] RestartScript()
        {
            Return "Import-Module FightingEntropy",
            '$Path            = Get-ChildItem "$Env:ProgramData\Secure Digits Plus LLC\FEDCPromo" | %
Fullname',
            '$InputObject     = Get-Content $Path\InputObject.json | ConvertFrom-Json',
            '$Adds            = @{ }',
            '$Adds.Credential = Import-CliXml $Path\Credential.txt',
            '$Adds.SafeModeAdministratorPassword = Import-CliXml $Path\Dsrm.txt',
            'ForEach ($Name in $InputObject.PSObject.Properties.Name)',
            '{ ',
            '    If ($Name.Length -gt 0 -and $Name -notin "Credential","SafeModeAdministratorPassword")',
            '    { ',
            '        $Adds.Add($Name,$InputObject.$Name)',
            '    }',
            '}',
            'Get-ChildItem $Path | Remove-Item -Verbose',
            'Unregister-ScheduledTask -TaskName FEDCPromo -Confirm:$False',
            'Get-Process -Name ServerManager -EA 0 | Stop-Process -EA 0',
            'Get-FEDCPromo -InputObject $Adds'
        }
        [Object] ScheduledTaskAction()
        {
            $Path     = "$Env:ProgramData\Secure Digits Plus LLC\FEDCPromo"
            $Command  = "Import-Module FightingEntropy;& '$Path\script.ps1'"
            $Argument = "-NoExit -ExecutionPolicy Bypass -Command `"$Command`""

            Return New-ScheduledTaskAction -Execute powershell -Argument $Argument
        }
        [Object] ScheduledTaskTrigger()
        {
```

```
        Return New-ScheduledTaskTrigger -AtLogon
    }
    [Void] RegisterScheduledTask()
    {
        $Splat = @{

            Action      = $This.ScheduledTaskAction()
            Trigger     = $This.ScheduledTaskTrigger()
            TaskName    = "FEDCPromo"
            RunLevel    = "Highest"
            Description = "Restart, then promote the system"
        }

        Register-ScheduledTask @Splat -Verbose
    }
    Execute()
    {
        # <MUST WRITE RULES HERE FOR INPUTOBJECT>

        # [Clear/Install Features]
        $This.Update(0,"Clearing [~] Execution Feature list")
        $This.Execution.Clear("Feature")

        $This.Update(0,"Clearing [~] Execution Output table")
        $This.Execution.Clear("Output")

        ForEach ($Item in $This.Feature.Output | ? Enable | ? Install)
        {
            $This.Update(1,"Adding [~] Execution Feature: [$($Item.Name)]")
            $This.Execution.Feature += $Item
        }

        If ($This.Control.Mode -in 1,2)
        {
            $Value = $This.Control.Current("DomainType").Value
            $This.Update(1,"Adding [~] Execution Output [Name: 'DomainType', Value: '$Value']")
            $This.Execute.Output.Add("DomainType",$Value)
        }

        # [Profile Items]
        ForEach ($Item in $This.Control.Profile.List("Slot") | ? IsEnabled)
        {
            $Name  = $Item.Name
            $Value = $Item.Value
            $This.Update(1,"Adding [~] Execution Output [Name: '$Name', Value: '$Value']")
            $This.Execution.Output.Add($Name,$Value)
        }

        # [Profile Roles]
        ForEach ($Item in $This.Control.Profile.List("Role") | ? IsEnabled)
        {
            $Name  = $Item.Name
            $Value = $Item.IsChecked
            $This.Update(1,"Adding [~] Execution Output [Name: '$Name', Value: '$Value']")
            $This.Execution.Output.Add($Name,$Value)
        }

        # [Database/Sysvol/Log Paths]
        ForEach ($Name in "DatabasePath","SysvolPath","LogPath")
        {
            $Value = $This.Xaml.Get($Name).Text
            $This.Update(1,"Adding [~] Execution Output [Name: '$Name', Value: '$Value']")
            $This.Execution.Output.Add($Name,$Value)
        }

        # [DSRM/Password]
        $Name  = "SafeModeAdministratorPassword"
        $Value = $This.Dsrm("Password").Value
        $This.Update(1,"Adding [~] Execution Output [Name: '$Name', Value: '$Value']")
        $This.Execution.Output.Add($Name,$Value)

        # [Credential]
```

```
            If ($This.Credential)
            {
                $This.Update(1,"Adding [~] Execution Output [Name: 'Credential', Value: '$
($This.Credential)']")
                $This.Execution.Output.Add("Credential",$This.Credential)
            }

            If ($This.Execution.Output["ReplicationSourceDC"] -eq "<Any>")
            {
                $This.Execution.Output["ReplicationSourceDC"] = $Null
            }

            If ($This.Execution.Output)
            {
                $This.ExportFile("Feature",$This.Execution.Feature)
                $This.ExportFile("Control",$This.Execution.Output)
            }

            $Splat = $This.Execution.Output
            Switch ($This.Test)
            {
                0
                {
                    $This.Update(0,"Installing [~] [FightingEntropy($([Char]960))] FEDCPromo -> Feature
installation")
                    $This.Module.Write($This.Console.Last().Status)

                    If ($This.Execution.Feature)
                    {
                        $This.Execution.Clear("Result")

                        ForEach ($Item in $This.Execution.Feature)
                        {
                            If ($Item.Name -notmatch "DNS")
                            {
                                $This.Update(0,"Installing [~] Name: [$($Item.Name)]")

                                $This.Execution.Result += $This.InstallWindowsFeature($Item.Name)

                                $This.Update(1,"Installed [+] Name: [$($Item.Name)]")
                            }
                        }
                    }

                    If (($This.Execution.Result | ? RestartNeeded -eq No).Count -gt 0)
                    {
                        $This.Update(0,"Reboot [!] required to proceed")
                        $This.Module.Write($This.Console.Last().Status)
                        <# INPUTOBJECT LOGIC HERE
                        If ($InputObject)
                        {
                            $This.ExportFile("InputObject",$InputObject)
                            $This.ExportFile( "Credential",$InputObject.Credential)
                            $This.ExportFile(       "Dsrm",$InputObject.SafeModeAdministratorPassword)
                            $This.ExportFile(     "Script",$This.RestartScript())
                            $This.RegisterScheduledTask()

                            $This.Update(0,"Restarting [~] $Env:ComputerName")
                            $This.Module.Write($This.Console.Last().Status)

                            If ($This.Mode -eq 1)
                            {
                                $This.DumpConsole()
                            }

                            $X = 5
                            Do
                            {
                                Write-Host $X
                                Start-Sleep 1
                                $X --
                            }
```

```
                            Until ($X -eq 0)
                            Restart-Computer
                        }
                        If (!$InputObject)
                        {
                            Throw "Write logic here"
                        }
                        #>
                    }
                    If (($This.Execution.Result | ? RestartNeeded -eq No).Count -eq 0)
                    {
                        $This.Update(0,"Installing [~] [FightingEntropy($([char]960))] Domain Controller")
                        $This.Module.Write($This.Console.Last().Status)

                        Switch ($This.Command.Slot)
                        {
                            {$_ -eq 0}
                            {
                                $This.InstallDomainController("AddsForest",$Splat)
                            }
                            {$_ -in 1,2}
                            {
                                $This.InstallDomainController("AddsDomain",$Splat)
                            }
                            {$_ -eq 3}
                            {
                                $This.InstallDomainController("AddsDomainController",$Splat)
                            }
                        }
                    }
                }
                1
                {
                    $This.Update(0,"Testing [~] [FightingEntropy($([Char]960))] FEDCPromo -> Feature
installation")
                    $This.Module.Write($This.Console.Last().Status)

                    ForEach ($Item in $This.Execution.Feature)
                    {
                        $This.Update(0,"Command [~] Install-WindowsFeature -Name $($Item.Name)
-IncludeAllSubFeature -IncludeManagementTools")
                    }

                    $This.Update(0,"Testing [~] [FightingEntropy($([Char]960))] FEDCPromo -> Test [$
($This.Control.Profile.Name)]")
                    $This.Module.Write($This.Console.Last().Status)

                    Switch ($This.Control.Slot)
                    {
                        {$_ -eq 0}
                        {
                            $This.InstallDomainController("TestAddsForest",$Splat)
                        }
                        {$_ -in 1,2}
                        {
                            $This.InstallDomainController("TestAddsDomain",$Splat)
                        }
                        {$_ -eq 3}
                        {
                            $This.InstallDomainController("TestAddsDomainController",$Splat)
                        }
                    }
                }
            }
        }
        StageXaml()
        {
            $Ctrl          = $This

            # // ===========
            # // | Command |
            # // ===========
```

```powershell
        # [ComboBox] Command slot
        $Ctrl.Reset($Ctrl.Xaml.IO.CommandSlot,$Ctrl.Control.Command.Output.Type)

        # [ComboBox] Command slot (Event handler)
        $Ctrl.Xaml.IO.CommandSlot.Add_SelectionChanged(
        {
            $Index = $Ctrl.Xaml.IO.CommandSlot.SelectedIndex
            $Ctrl.SetProfile($Index)
            $Ctrl.Reset($Ctrl.Xaml.IO.Command,$Ctrl.Control.Profile)
        })

        # // ========
        # // | Mode |
        # // ========

        # [DataGrid] OS Caption
        $Ctrl.Reset($Ctrl.Xaml.IO.OperatingSystemCaption,$Ctrl.System.OperatingSystem)

        # [Datagrid] OS Properties
        $Ctrl.Reset($Ctrl.Xaml.IO.OperatingSystemExtension,$Ctrl.System.OperatingSystem)

        # [ComboBox] Forest mode
        $Ctrl.Reset($Ctrl.Xaml.IO.ForestMode,$Ctrl.Control.ForestMode.Output)

        # [ComboBox -> Ctrl.Control -> Datagrid] Forest mode (Event handler)
        $Ctrl.Xaml.IO.ForestMode.Add_SelectionChanged(
        {
            $Ctrl.Control.ForestMode.Selected = $Ctrl.Xaml.IO.ForestMode.SelectedIndex
            $Ctrl.Reset($Ctrl.Xaml.IO.ForestModeExtension,$Ctrl.Control.Current("ForestMode"))
        })

        # [ComboBox] Forest Mode (Default)
        $Ctrl.Xaml.IO.ForestMode.SelectedIndex = ($Ctrl.Control.ForestMode.Output | ? Enable)[0].Index

        # [ComboBox] Domain mode
        $Ctrl.Reset($Ctrl.Xaml.IO.DomainMode,$Ctrl.Control.DomainMode.Output)

        # [ComboBox -> Ctrl.Control -> Datagrid] Domain mode (Event handler)
        $Ctrl.Xaml.IO.DomainMode.Add_SelectionChanged(
        {
            $Ctrl.Control.DomainMode.Selected = $Ctrl.Xaml.IO.DomainMode.SelectedIndex
            $Ctrl.Reset($Ctrl.Xaml.IO.DomainModeExtension,$Ctrl.Control.Current("DomainMode"))
        })

        # [ComboBox] Domain mode (Default)
        $Ctrl.Xaml.IO.DomainMode.SelectedIndex = ($Ctrl.Control.DomainMode.Output | ? Enable)[0].Index

        # [ComboBox] Replication Source DC
        $Ctrl.Reset($Ctrl.Xaml.IO.ReplicationSourceDC,$Null)

        # // ============
        # // | Features |
        # // ============

        # [DataGrid] Windows Features
        $Ctrl.Reset($Ctrl.Xaml.IO.Feature,$Ctrl.Feature.Output)

        # // ===============
        # // | Roles/Paths |
        # // ===============

        # [CheckBox[]] Roles
        $Ctrl.XamlDefault("Role")

        # [CheckBox] Install Dns (Event handler)
        $Ctrl.Xaml.IO.InstallDNS.Add_IsEnabledChanged(
        {
            $Ctrl.Role("InstallDns").Toggle()
        })

        # [CheckBox] Create Dns Delegation (Event handler)
```

```powershell
$Ctrl.Xaml.IO.CreateDnsDelegation.Add_IsEnabledChanged(
{
    $Ctrl.Role("CreateDnsDelegation").Toggle()
})

# [CheckBox] No Global Catalog (Event handler)
$Ctrl.Xaml.IO.NoGlobalCatalog.Add_IsEnabledChanged(
{
    $Ctrl.Role("NoGlobalCatalog").Toggle()
})

# [CheckBox] Critical Replication Only (Event handler)
$Ctrl.Xaml.IO.CriticalReplicationOnly.Add_IsEnabledChanged(
{
    $Ctrl.Role("CriticalReplicationOnly").Toggle()
})

# [TextBox[]] DatabasePath, SysvolPath, Logpath
$Ctrl.XamlDefault("Path")

# [Button] Database path (Event handler)
$Ctrl.Xaml.IO.DatabaseBrowse.Add_Click(
{
    $Ctrl.Browse("DatabasePath")
})

# [Button] Sysvol path (Event handler)
$Ctrl.Xaml.IO.SysvolBrowse.Add_Click(
{
    $Ctrl.Browse("SysvolPath")
})

# [Button] Log path (Event handler)
$Ctrl.Xaml.IO.LogBrowse.Add_Click(
{
    $Ctrl.Browse("LogPath")
})

# // =========
# // | Names |
# // =========

# [TextBox[]] Domain Names
$Ctrl.XamlDefault("Name")

# [Button -> TextBox] Credential
$Ctrl.XamlDefault("Credential")

# [Button[]] Start, Cancel
$Ctrl.XamlDefault("Button")

# [TextBox] ParentDomainName (Text changed event handler)
$Ctrl.Xaml.IO.ParentDomainName.Add_TextChanged(
{
    $Ctrl.Check("ParentDomainName")
})

# [TextBox] ParentDomainName (Got focus event handler)
$Ctrl.Xaml.IO.ParentDomainName.Add_GotFocus(
{
    If ($Ctrl.Xaml.IO.ParentDomainName.Text -eq $Ctrl.DefaultText("ParentDomainName"))
    {
        $Ctrl.ToggleStaging()
        $Ctrl.Xaml.IO.ParentDomainName.Text = ""
        $Ctrl.ToggleStaging()
    }
})

# [TextBox] ParentDomainName (Lost focus event handler)
$Ctrl.Xaml.IO.ParentDomainName.Add_LostFocus(
{
    If ($Ctrl.Xaml.IO.ParentDomainName.Text -eq "")
```

```
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.ParentDomainName.Text = $Ctrl.DefaultText("ParentDomainName")
            $Ctrl.ToggleStaging()
        }
    })

    # [TextBox] DomainName (Text changed event handler)
    $Ctrl.Xaml.IO.DomainName.Add_TextChanged(
    {
        $Ctrl.Check("DomainName")
    })

    # [TextBox] DomainName (Got focus event handler)
    $Ctrl.Xaml.IO.DomainName.Add_GotFocus(
    {
        If ($Ctrl.Xaml.IO.DomainName.Text -eq $Ctrl.DefaultText("DomainName"))
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.DomainName.Text = ""
            $Ctrl.ToggleStaging()
        }
    })

    # [TextBox] DomainName (Lost focus event handler)
    $Ctrl.Xaml.IO.DomainName.Add_LostFocus(
    {
        If ($Ctrl.Xaml.IO.DomainName.Text -eq "")
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.DomainName.Text = $Ctrl.DefaultText("DomainName")
            $Ctrl.ToggleStaging()
        }
    })

    # [TextBox] NewDomainName (Event handler)
    $Ctrl.Xaml.IO.NewDomainName.Add_TextChanged(
    {
        $Ctrl.Check("NewDomainName")
    })

    # [TextBox] NewDomainName (Got focus event handler)
    $Ctrl.Xaml.IO.NewDomainName.Add_GotFocus(
    {
        If ($Ctrl.Xaml.IO.NewDomainName.Text -eq $Ctrl.DefaultText("NewDomainName"))
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.NewDomainName.Text = ""
            $Ctrl.ToggleStaging()
        }
    })

    # [TextBox] NewDomainName (Lost focus event handler)
    $Ctrl.Xaml.IO.NewDomainName.Add_LostFocus(
    {
        If ($Ctrl.Xaml.IO.NewDomainName.Text -eq "")
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.NewDomainName.Text = $Ctrl.DefaultText("NewDomainName")
            $Ctrl.ToggleStaging()
        }
    })

    # [TextBox] DomainNetBiosName (Event handler)
    $Ctrl.Xaml.IO.DomainNetBIOSName.Add_TextChanged(
    {
        $Ctrl.Check("DomainNetBiosName")
    })

    # [TextBox] DomainNetBiosName (Got focus event handler)
    $Ctrl.Xaml.IO.DomainNetBiosName.Add_GotFocus(
    {
```

```
        If ($Ctrl.Xaml.IO.DomainNetBiosName.Text -eq $Ctrl.DefaultText("DomainNetBiosName"))
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.DomainNetBiosName.Text = ""
            $Ctrl.ToggleStaging()
        }
    })

    # [TextBox] DomainNetBiosName (Lost focus event handler)
    $Ctrl.Xaml.IO.DomainNetBiosName.Add_LostFocus(
    {
        If ($Ctrl.Xaml.IO.DomainNetBiosName.Text -eq "")
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.DomainNetBiosName.Text = $Ctrl.DefaultText("DomainNetBiosName")
            $Ctrl.ToggleStaging()
        }
    })

    # [TextBox] NewDomainNetBiosName (Event handler)
    $Ctrl.Xaml.IO.NewDomainNetBiosName.Add_TextChanged(
    {
        $Ctrl.Check("NewDomainNetBiosName")
    })

    # [TextBox] NewDomainNetBiosName (Got focus event handler)
    $Ctrl.Xaml.IO.NewDomainNetBiosName.Add_GotFocus(
    {
        If ($Ctrl.Xaml.IO.NewDomainNetBiosName.Text -eq $Ctrl.DefaultText("NewDomainNetBiosName"))
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.NewDomainNetBiosName.Text = ""
            $Ctrl.ToggleStaging()
        }
    })

    # [TextBox] NewDomainNetBiosName (Lost focus event handler)
    $Ctrl.Xaml.IO.NewDomainNetBiosName.Add_LostFocus(
    {
        If ($Ctrl.Xaml.IO.NewDomainNetBiosName.Text -eq "")
        {
            $Ctrl.ToggleStaging()
            $Ctrl.Xaml.IO.NewDomainNetBiosName.Text = $Ctrl.DefaultText("NewDomainNetBiosName")
            $Ctrl.ToggleStaging()
        }
    })

    # // ==============
    # // | Credential |
    # // ==============

    # [Button] Login (Event handler)
    $Ctrl.Xaml.IO.CredentialButton.Add_Click(
    {
        $Ctrl.Login()
    })

    # [TextBox] Dsrm Password (Event handler)
    $Ctrl.Xaml.IO.SafeModeAdministratorPassword.Add_PasswordChanged(
    {
        $Name                       = "SafeModeAdministratorPassword"
        $Icon                       = "{0}Icon" -f $Name

        $Pass                       = $Ctrl.Dsrm("Password")
        $Pass.Value                 = $Ctrl.Xaml.IO.$Name.Password
        $Ctrl.CheckDSRMPassword()
        $Pass                       = $Ctrl.Dsrm("Password")

        $Ctrl.Xaml.IO.$Icon.Source  = $Ctrl.Icon($Pass.Check)
        $Ctrl.Xaml.IO.$Icon.Tooltip = $Pass.Reason
        $Ctrl.Total()
    })
```

```
        # [TextBox] Dsrm Confirm (Event handler)
        $Ctrl.Xaml.IO.Confirm.Add_PasswordChanged(
        {
            $Name                          = "Confirm"
            $Icon                          = "{0}Icon" -f $Name

            $Pass                          = $Ctrl.Dsrm("Confirm")
            $Pass.Value                    = $Ctrl.Xaml.IO.$Name.Password
            $Ctrl.CheckDSRMConfirm()
            $Pass                          = $Ctrl.Dsrm("Confirm")

            $Ctrl.Xaml.IO.$Icon.Tooltip    = $Pass.Reason
            $Ctrl.Xaml.IO.$Icon.Source     = $Ctrl.Icon($Pass.Check)
            $Ctrl.Total()
        })

        # // ==========
        # // | Bottom |
        # // ==========

        # [Button] Start (Event handler)
        $Ctrl.Xaml.IO.Start.Add_Click(
        {
            $Ctrl.Test = 0
            $Ctrl.Complete()
            $Ctrl.Xaml.IO.DialogResult = 1
        })

        # [Button] Test (Event handler)
        $Ctrl.Xaml.IO.Test.Add_Click(
        {
            $Ctrl.Test = 1
            $Ctrl.Complete()
            $Ctrl.Xaml.IO.DialogResult = 1
        })

        # [Button] Cancel (Event handler)
        $Ctrl.Xaml.IO.Cancel.Add_Click(
        {
            $Ctrl.Xaml.IO.DialogResult = 0
        })

        $Ctrl.Xaml.IO.CommandSlot.SelectedIndex = 0
        $Ctrl.SetProfile(0)
    }
}
```

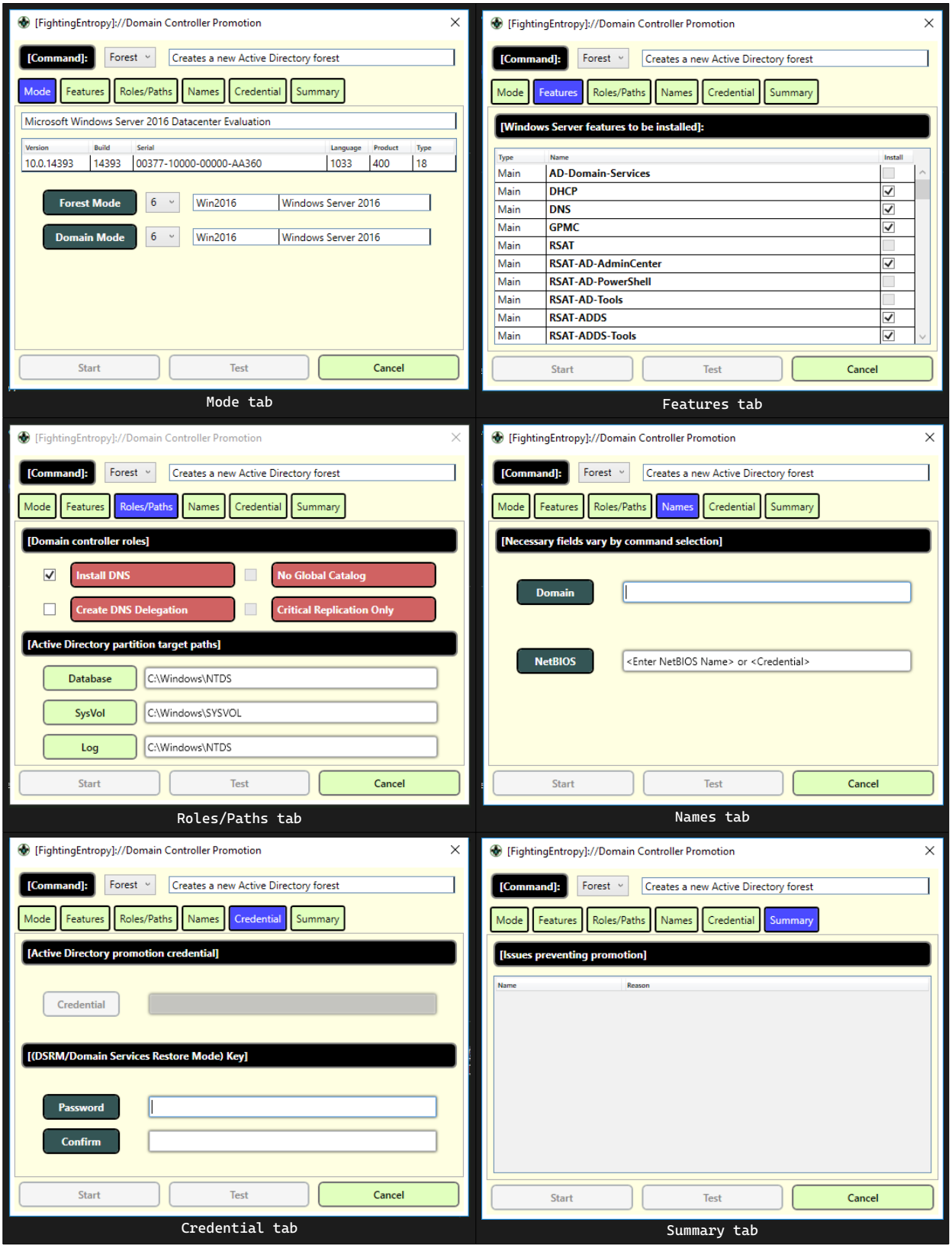
```
                                                                                           ______________________________/
\_________________________________________________________________________________________/ Class [FEDCPromoController]
  Output /--------------------------------------------------------------------------------------------------------------\
/--------
```

The above information can be accessed when the function wrapper is left out of the equation...

```
PS Prompt:\> $Mode = 1
PS Prompt:\> $Ctrl = [FEDCPromoController]::New($Mode)
PS Prompt:\> $Ctrl.Xaml.Invoke()

[00:00:00] (State: 0/Status: Running [~] (1/3/2023 4:54:26 PM))
[00:00:00.9102314] (State: 1/Status: Loaded [+] [FEFeatureList])
[00:00:00.9132337] (State: 1/Status: Valid [+] Module: [AddsDeployment])
[00:00:00.9162348] (State: 0/Status: Importing [~] Module: [AddsDeployment])
[00:00:01.4833791] (State: 1/Status: Loaded [+] [FEModule])
[00:00:08.1534227] (State: 1/Status: Loaded [+] [FESystem])
[00:00:28.3766022] (State: 1/Status: Loaded [+] [FENetwork])
[00:00:28.4276158] (State: 0/Status: Detected [!] Current system is a virtual machine)
    ____    ____________________________________________________________________________________________
   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
   \\__//¯¯¯ Detected [!] Current system is a virtual machine                                        ___//¯¯\\
    ¯¯¯\\__________________________________________________________________________________________//¯¯\\__//
        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
```

```
[00:00:31.6614412] (State: 1/Status: Loaded [+] [CommandController])
[00:00:31.7904785] (State: 1/Status: Loaded [+] [ValidationController])
[00:00:31.9025048] (State: 1/Status: Loaded [+] [ExecutionController])
[00:00:32.9506336] (State: 1/Status: Loaded [+] [FEDCPromoXaml])
[00:00:32.9916516] (State: 0/Status: Resetting [~] Xaml.IO.CommandSlot [ComboBox])
[00:00:33.0876670] (State: 0/Status: Resetting [~] Xaml.IO.OperatingSystemCaption [DataGrid])
[00:00:33.1446766] (State: 0/Status: Resetting [~] Xaml.IO.OperatingSystemExtension [DataGrid])
[00:00:33.1776880] (State: 0/Status: Resetting [~] Xaml.IO.ForestMode [ComboBox])
[00:00:33.4157484] (State: 0/Status: Resetting [~] Xaml.IO.ForestModeExtension [DataGrid])
[00:00:33.4207694] (State: 0/Status: Resetting [~] Xaml.IO.DomainMode [ComboBox])
[00:00:33.4437550] (State: 0/Status: Resetting [~] Xaml.IO.DomainModeExtension [DataGrid])
[00:00:33.4487560] (State: 0/Status: Resetting [~] Xaml.IO.ReplicationSourceDC [ComboBox])
[00:00:33.4547615] (State: 0/Status: Resetting [~] Xaml.IO.Feature [DataGrid])
[00:00:33.4617603] (State: 0/Status: Setting [~] Defaults: [Role])
[00:00:33.5037721] (State: 0/Status: Setting [~] Defaults: [Path])
[00:00:33.5827904] (State: 0/Status: Setting [~] Defaults: [Name])
[00:00:33.6708142] (State: 0/Status: Setting [~] Defaults: [Credential])
[00:00:33.6908206] (State: 0/Status: Setting [~] Defaults: [Button])
[00:00:33.8198725] (State: 0/Status: Changed [+] Selection [0])
[00:00:33.9498888] (State: 0/Status: Setting [~] DomainType)
[00:00:34.0699167] (State: 0/Status: Setting [~] Credential button)
[00:00:34.1279316] (State: 0/Status: Settings [~] Features)
[00:00:34.1449347] (State: 0/Status: Setting [~] Roles)
[00:00:34.1739432] (State: 0/Status: Setting [~] Role [X], Name: [InstallDns], Enabled: [1], Checked: [1])
[00:00:34.2199533] (State: 0/Status: Setting [~] Role [X], Name: [CreateDnsDelegation], Enabled: [1], Checked: [0])
[00:00:34.2529625] (State: 0/Status: Setting [~] Role [ ], Name: [CriticalReplicationOnly], Enabled: [0], Checked: [0])
[00:00:34.2959745] (State: 0/Status: Setting [~] Role [ ], Name: [NoGlobalCatalog], Enabled: [0], Checked: [0])
[00:00:34.3649896] (State: 0/Status: Setting [~] Profile slot)
[00:00:34.4100021] (State: 0/Status: Setting [~] Profile [X], Slot: [ForestMode], Type: [ComboBox])
[00:00:34.4680168] (State: 0/Status: Setting [~] Profile [X], Slot: [DomainMode], Type: [ComboBox])
[00:00:34.4820270] (State: 0/Status: Setting [~] Profile [ ], Slot: [ReplicationSourceDC], Type: [ComboBox])
[00:00:34.5040275] (State: 0/Status: Setting [~] Profile [ ], Slot: [SiteName], Type: [ComboBox])
[00:00:34.5130273] (State: 0/Status: Setting [~] Profile [ ], Slot: [ParentDomainName], Type: [TextBox])
[00:00:34.5660441] (State: 0/Status: Setting [~] Profile [X], Slot: [DomainName], Type: [TextBox])
[00:00:34.6440625] (State: 0/Status: Setting [~] Profile [X], Slot: [DomainNetBIOSName], Type: [TextBox])
[00:00:34.6800772] (State: 0/Status: Setting [~] Profile [ ], Slot: [NewDomainName], Type: [TextBox])
[00:00:34.7290821] (State: 0/Status: Setting [~] Profile [ ], Slot: [NewDomainNetBIOSName], Type: [TextBox])
[00:00:34.7571394] (State: 0/Status: Setting [~] Connection)
[00:00:34.7630936] (State: 0/Status: Clearing [~] Credential)
[00:00:34.7700949] (State: 0/Status: Setting [~] DSRM password)
[00:00:34.8121049] (State: 0/Status: Resetting [~] Xaml.IO.Command [DataGrid])
[00:00:34.8401209] (State: 0/Status: Changed [+] Selection [0])
[00:00:34.8921270] (State: 0/Status: Setting [~] DomainType)
[00:00:34.9301368] (State: 0/Status: Setting [~] Credential button)
[00:00:34.9321412] (State: 0/Status: Settings [~] Features)
[00:00:34.9331428] (State: 0/Status: Setting [~] Roles)
[00:00:34.9371387] (State: 0/Status: Setting [~] Role [X], Name: [InstallDns], Enabled: [1], Checked: [1])
[00:00:34.9401408] (State: 0/Status: Setting [~] Role [X], Name: [CreateDnsDelegation], Enabled: [1], Checked: [0])
[00:00:34.9411410] (State: 0/Status: Setting [~] Role [ ], Name: [CriticalReplicationOnly], Enabled: [0], Checked: [0])
[00:00:34.9421402] (State: 0/Status: Setting [~] Role [ ], Name: [NoGlobalCatalog], Enabled: [0], Checked: [0])
[00:00:34.9431409] (State: 0/Status: Setting [~] Profile slot)
[00:00:34.9611455] (State: 0/Status: Setting [~] Profile [X], Slot: [ForestMode], Type: [ComboBox])
[00:00:34.9621445] (State: 0/Status: Setting [~] Profile [X], Slot: [DomainMode], Type: [ComboBox])
[00:00:34.9621445] (State: 0/Status: Setting [~] Profile [ ], Slot: [ReplicationSourceDC], Type: [ComboBox])
[00:00:34.9631444] (State: 0/Status: Setting [~] Profile [ ], Slot: [SiteName], Type: [ComboBox])
[00:00:34.9641460] (State: 0/Status: Setting [~] Profile [ ], Slot: [ParentDomainName], Type: [TextBox])
[00:00:34.9641460] (State: 0/Status: Setting [~] Profile [X], Slot: [DomainName], Type: [TextBox])
[00:00:34.9651449] (State: 0/Status: Setting [~] Profile [X], Slot: [DomainNetBIOSName], Type: [TextBox])
[00:00:34.9651449] (State: 0/Status: Setting [~] Profile [ ], Slot: [NewDomainName], Type: [TextBox])
[00:00:34.9661457] (State: 0/Status: Setting [~] Profile [ ], Slot: [NewDomainNetBIOSName], Type: [TextBox])
[00:00:34.9661457] (State: 0/Status: Setting [~] Connection)
[00:00:34.9661457] (State: 0/Status: Clearing [~] Credential)
[00:00:34.9681449] (State: 0/Status: Setting [~] DSRM password)
```

At which point, the following pictures represent what all of that above stuff actually DOES...
(Click on the picture to expand)



Now, the information in each of these pictures is just showing the starting position of the command “Forest”.

The truth is, it RESPONDS to the INPUT, and I'll demonstrate that in the next series of pictures.

But first, I'd like to cover some of the code-behind that illustrates the console output.

```
PS Prompt:\> $Ctrl

Console    : 00:26:47.8496821
Mode       : 1
Staging    : 0
Test       : 0
Xaml       : <FightingEntropy.XamlWindow>
Module     : <FightingEntropy.Module.Controller>
System     : SERVER01, Microsoft Corporation | Virtual Machine, Microsoft Windows Server 2016 Datacenter
Evaluation 10.0.14393-14393
Network    : <FENetwork.NetworkControllerMaster>
Caption    : Microsoft Windows Server 2016 Datacenter Evaluation
Server     : 6
Feature    : <FEDCPromo.FeatureController>
Control    : <FEDCPromo.CommandController>
Connection :
Credential :
Validation : (80) <FEDCPromo.ValidationController>
Execution  : <FEDCPromo.ExecutionController>

PS Prompt:\>
```

This is the main controller class, and many of these properties have subproperties, submethods, subconstructors, substrings, and various other odds and ends that I haven't specifically mentioned.

The console property isn't just a stopwatch...

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

Start  : 1/3/2023 4:54:26 PM
End    : 1/1/0001 12:00:00 AM
Span   :
Status : [00:09:40.8817990] (State: 0/Status: Resetting [~] Xaml.IO.Command [DataGrid])
Output : {[00:00:00] (State: 0/Status: Running [~] (1/3/2023 4:54:26 PM)),
          [00:00:00.9102314] (State: 1/Status: Loaded [+] [FEFeatureList]),
          [00:00:00.9132337] (State: 1/Status: Valid [+] Module: [AddsDeployment]),
          [00:00:00.9162348] (State: 0/Status: Importing [~] Module: [AddsDeployment])...}

PS Prompt:\>
```

Probably looks a little familiar. That's because this thing provides the console output above.

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

Names     : {Border, CommandSlot, Command, OperatingSystemCaption...}
Types     : {CommandSlot, Command, OperatingSystemCaption, OperatingSystemExtension...}
Node      : System.Xml.XmlNodeReader
IO        : System.Windows.Window
Exception :

PS Prompt:\>
```

That probably looks a little familiar if the reader has read the previous PDF files that I've written to cover other utilities of the module.

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

Source      : https://www.github.com/mcc85s/FightingEntropy
Name        : [FightingEntropy(π)]
Description : Beginning the fight against ID theft and cybercrime
Author      : Michael C. Cook Sr.
Company     : Secure Digits Plus LLC
Copyright   : (c) 2022 (mcc85s/mcc85sx/sdp). All rights reserved.
Guid        : 0b36cfa4-dfad-4863-9171-f8afe65769cf
Date        : 11/7/2022 4:01:21 PM
Version     : 2022.12.0
OS          : <FightingEntropy.Module.OS>
Root        : <FightingEntropy.Module.Root>
Manifest    : <FightingEntropy.Module.Manifest>
```

```
Registry    : <FightingEntropy.Module.RegistryKey>
System      :

PS Prompt:\>
```

That'll probably look familiar if anybody has actually been interested in the development of this module. Because this is all stuff that I feature in this video...

Date	Name	Url
10/28/22	[FightingEntropy(π)][2022.10.1]	https://youtu.be/S7k4lZdPE-I

I'm not going to cover every single subproperty here, because I would imagine that if the software engineers at Microsoft (or wherever else, really) had to cover every aspect or (subproperty/method/function) of the things THEY work on, every single time they had to document something they were working on...?

Uh- they'd literally have to cover the same things over and over again, and spend most of their life covering things they already covered and hadn't changed. Not a real great use of time, to cover stuff that hasn't been updated or changed.

That wouldn't be the case if they made changes to some stuff after all.

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


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

PS Prompt:\>
```

So, I covered this utility rather recently. I've made some changes to it so that it can capture the input I'd like to see more readily. This will eventually cover other aspects that aren't seen here. As I mentioned above, I'm not going to cover every single subproperty, method, or whatever here, because I haven't changed much since the last document I wrote specifically detailing “Get-FESystem”.

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

Mode        : 7
Class       : (256) <FENetwork.V4ClassList>
Vendor      : (28664) <FENetwork.VendorList>
Arp         : (1) <FENetwork.ArpList>
Nbt         : (1) <FENetwork.NbtStatList>
NetStat     : (50) <FENetwork.NetStatList>
Adapter     : (14) <FENetwork.NetworkAdapterList>
Config      : (14) <FENetwork.NetworkAdapterConfigList>
Route       : (23) <FENetwork.NetworkRouteList>
Interface   : (6) <FENetwork.NetworkInterfaceList>
Ip          : (7) <FENetwork.NetworkIpList>
Compartment : (0) <FENetwork.NetworkControllerCompartmentList>

PS Prompt:\>
```

I've also covered this utility rather recently, Get-FESystem.

Of the last (3) things I just covered up above, they do a lot of the heavy lifting in terms of obtaining (module/system/network) details, and then preparing those details in a way where there are EFFICIENCY shortcuts.

Such as being able to use a pipeline symbol, question mark, and then a property name... and in some cases maybe using like a comparison operator like (-eq/-ne/-gt/-lt) and then another property name or regex match.

You'd be amazed as to how elaborate this [PowerShell] language truly is when you're an expert at using it.

```
PS Prompt:\> $Ctrl.Caption
Microsoft Windows Server 2016 Datacenter Evaluation
PS Prompt:\>
```

That is a string. There's no additional properties for that one.

```
PS Prompt:\> $Ctrl.Server
6
PS Prompt:\>
```

That is an integer that is used to select the item in the ComboBox list for (ForestMode/DomainMode)

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

Name    Output
----    ------
Feature {AD-Domain-Services, DHCP, DNS, GPMC...}


PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Feature.Output | Format-Table

Index Type     Name                       State Enable Install
----- ----     ----                       ----- ------ -------
    0 Main     AD-Domain-Services             1      0       0
    1 Main     DHCP                           0      1       1
    2 Main     DNS                            0      1       1
    3 Main     GPMC                           0      1       1
    4 Main     RSAT                           1      0       0
    5 Main     RSAT-AD-AdminCenter            0      1       1
    6 Main     RSAT-AD-PowerShell             1      0       0
    7 Main     RSAT-AD-Tools                  1      0       0
    8 Main     RSAT-ADDS                      0      1       1
    9 Main     RSAT-ADDS-Tools                0      1       1
   10 Main     RSAT-DHCP                      0      1       1
   11 Main     RSAT-DNS-Server                0      1       1
   12 Main     RSAT-Role-Tools                1      0       0
   13 WDS      WDS                            0      1       1
   14 WDS      WDS-AdminPack                  0      1       1
   15 WDS      WDS-Deployment                 0      1       1
   16 WDS      WDS-Transport                  0      1       1
   17 IIS      BITS                           0      1       1
   18 IIS      BITS-IIS-Ext                   0      1       1
   19 IIS      DSC-Service                    0      1       1
   20 IIS      FS-SMBBW                       0      1       1
   21 IIS      ManagementOData                0      1       1
   22 IIS      Net-Framework-45-ASPNet        0      1       1
   23 IIS      Net-WCF-HTTP-Activation45      0      1       1
   24 IIS      RSAT-BITS-Server               0      1       1
   25 IIS      WAS                            0      1       1
   26 IIS      WAS-Config-APIs                0      1       1
   27 IIS      WAS-Process-Model              0      1       1
   28 IIS      WebDAV-Redirector              0      1       1
   29 IIS      Web-HTTP-Errors                0      1       1
   30 IIS      Web-HTTP-Logging               0      1       1
   31 IIS      Web-HTTP-Redirect              0      1       1
   32 IIS      Web-HTTP-Tracing               0      1       1
   33 IIS      Web-App-Dev                    0      1       1
   34 IIS      Web-AppInit                    0      1       1
   35 IIS      Web-Asp-Net45                  0      1       1
   36 IIS      Web-Basic-Auth                 0      1       1
   37 IIS      Web-Common-Http                0      1       1
   38 IIS      Web-Custom-Logging             0      1       1
   39 IIS      Web-DAV-Publishing             0      1       1
   40 IIS      Web-Default-Doc                0      1       1
   41 IIS      Web-Digest-Auth                0      1       1
   42 IIS      Web-Dir-Browsing               0      1       1
```

```
   43 IIS      Web-Filtering                0      1       1
   44 IIS      Web-Health                   0      1       1
   45 IIS      Web-Includes                 0      1       1
   46 IIS      Web-Log-Libraries            0      1       1
   47 IIS      Web-Metabase                 0      1       1
   48 IIS      Web-Mgmt-Console             0      1       1
   49 IIS      Web-Net-Ext45                0      1       1
   50 IIS      Web-Performance              0      1       1
   51 IIS      Web-Request-Monitor          0      1       1
   52 IIS      Web-Security                 0      1       1
   53 IIS      Web-Stat-Compression         0      1       1
   54 IIS      Web-Static-Content           0      1       1
   55 IIS      Web-Url-Auth                 0      1       1
   56 IIS      Web-WebServer                0      1       1
   57 IIS      Web-Windows-Auth             0      1       1
   58 IIS      Web-ISAPI-Ext                0      1       1
   59 IIS      Web-ISAPI-Filter             0      1       1
   60 IIS      Web-Server                   0      1       1
   61 IIS      WindowsPowerShellWebAccess   0      1       1
   62 Veridian Hyper-V                      0      0       0
   63 Veridian RSAT-Hyper-V-Tools           0      0       0
   64 Veridian Hyper-V-Tools                0      0       0
   65 Veridian Hyper-V-PowerShell           0      0       0

PS Prompt:\>
```

These are the features that are able to be seen in the features tab.

Some of these features will not be available if the machine is running and it is a virtual machine, particularly those with the type of “Veridian”. Veridian is the original code name for Hyper-V.

I'll eventually add support for servers that do NOT have the ability to run the virtualization hypervisor, but I think that doing so would be limiting the scope of how applicable this program is, as really, one of the main focuses of the module in general, is to provide control for both physical and virtual machines.

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

Name       : CommandController
Slot       : 0
Command    : <FEDCPromo.CommandTypeList>
DomainType : <FEDCPromo.DomainTypeList>
ForestMode : (9) <FEDCPromo.WindowsServerList[ForestMode]>
DomainMode : (9) <FEDCPromo.WindowsServerList[DomainMode]>
Profile    : <FEDCPromo.ProfileController[Forest]>

PS Prompt:\>
```

So, this is the command controller, and it has subproperties that are controllers as well.

```
PS Prompt:\> $Ctrl.Control.Command

Name    Selected Output
----    -------- ------
Command        0 {<FEDCPromo.CommandTypeItem>, <FEDCPromo.CommandTypeItem>, <FEDCPromo.CommandTypeItem>...}

PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Control.Command.Output

Index Type   Name                          Description
----- ----   ----                          -----------
    0 Forest Install-AddsForest            Creates a new Active Directory forest
    1 Tree   Install-AddsDomain            Creates a new Active Directory tree domain
    2 Child  Install-AddsDomain            Creates a new Active Directory child domain
    3 Clone  Install-AddsDomainController Adds a new domain controller to an existing domain

PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Control.DomainType

Name       Selected Output
----       -------- ------
```

```
DomainType        0 {Forest, Tree, Child, Clone}

PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Control.DomainType.Output

Index Name   Value
----- ----   -----
    0 Forest -
    1 Tree   Tree
    2 Child  Child
    3 Clone  -

PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Control.ForestMode

Name       Selected Output
----       -------- ------
ForestMode        0 {Windows Server 2000, Windows Server 2003, Windows Server 2008, Windows Server 2008 R2...}

PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Control.ForestMode.Output

Index Name      DisplayName            Enable
----- ----      -----------            ------
    0 Win2K     Windows Server 2000         0
    1 Win2003   Windows Server 2003         0
    2 Win2008   Windows Server 2008         0
    3 Win2008R2 Windows Server 2008 R2      0
    4 Win2012   Windows Server 2012         0
    5 Win2012R2 Windows Server 2012 R2      0
    6 Win2016   Windows Server 2016         1
    7 Win2019   Windows Server 2019         1
    8 Win2022   Windows Server 2022         1

PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Control.DomainMode

Name       Selected Output
----       -------- ------
DomainMode        6 {Windows Server 2000, Windows Server 2003, Windows Server 2008, Windows Server 2008 R2...}

PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Control.DomainMode.Output

Index Name      DisplayName            Enable
----- ----      -----------            ------
    0 Win2K     Windows Server 2000         0
    1 Win2003   Windows Server 2003         0
    2 Win2008   Windows Server 2008         0
    3 Win2008R2 Windows Server 2008 R2      0
    4 Win2012   Windows Server 2012         0
    5 Win2012R2 Windows Server 2012 R2      0
    6 Win2016   Windows Server 2016         1
    7 Win2019   Windows Server 2019         1
    8 Win2022   Windows Server 2022         1

PS Prompt:\>
```

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

Name       Output
----       ------
Validation {ANONYMOUS, AUTHENTICATED USER, BATCH, BUILTIN...}

PS Prompt:\>
```

```
PS Prompt:\> $Ctrl.Validation.Output

Index Type               Value
----- ----               -----
```

```
  0 Reserved           ANONYMOUS
  1 Reserved           AUTHENTICATED USER
  2 Reserved           BATCH
  3 Reserved           BUILTIN
  4 Reserved           CREATOR GROUP
  5 Reserved           CREATOR GROUP SERVER
  6 Reserved           CREATOR OWNER
  7 Reserved           CREATOR OWNER SERVER
  8 Reserved           DIALUP
  9 Reserved           DIGEST AUTH
 10 Reserved           INTERACTIVE
 11 Reserved           INTERNET
 12 Reserved           LOCAL
 13 Reserved           LOCAL SYSTEM
 14 Reserved           NETWORK
 15 Reserved           NETWORK SERVICE
 16 Reserved           NT AUTHORITY
 17 Reserved           NT DOMAIN
 18 Reserved           NTLM AUTH
 19 Reserved           NULL
 20 Reserved           PROXY
 21 Reserved           REMOTE INTERACTIVE
 22 Reserved           RESTRICTED
 23 Reserved           SCHANNEL AUTH
 24 Reserved           SELF
 25 Reserved           SERVER
 26 Reserved           SERVICE
 27 Reserved           SYSTEM
 28 Reserved           TERMINAL SERVER
 29 Reserved           THIS ORGANIZATION
 30 Reserved           USERS
 31 Reserved           WORLD
 32 Legacy             -GATEWAY
 33 Legacy             -GW
 34 Legacy             -TAC
 35 SecurityDescriptor AN
 36 SecurityDescriptor AO
 37 SecurityDescriptor AU
 38 SecurityDescriptor BA
 39 SecurityDescriptor BG
 40 SecurityDescriptor BO
 41 SecurityDescriptor BU
 42 SecurityDescriptor CA
 43 SecurityDescriptor CD
 44 SecurityDescriptor CG
 45 SecurityDescriptor CO
 46 SecurityDescriptor DA
 47 SecurityDescriptor DC
 48 SecurityDescriptor DD
 49 SecurityDescriptor DG
 50 SecurityDescriptor DU
 51 SecurityDescriptor EA
 52 SecurityDescriptor ED
 53 SecurityDescriptor HI
 54 SecurityDescriptor IU
 55 SecurityDescriptor LA
 56 SecurityDescriptor LG
 57 SecurityDescriptor LS
 58 SecurityDescriptor LW
 59 SecurityDescriptor ME
 60 SecurityDescriptor MU
 61 SecurityDescriptor NO
 62 SecurityDescriptor NS
 63 SecurityDescriptor NU
 64 SecurityDescriptor PA
 65 SecurityDescriptor PO
 66 SecurityDescriptor PS
 67 SecurityDescriptor PU
 68 SecurityDescriptor RC
 69 SecurityDescriptor RD
 70 SecurityDescriptor RE
 71 SecurityDescriptor RO
```

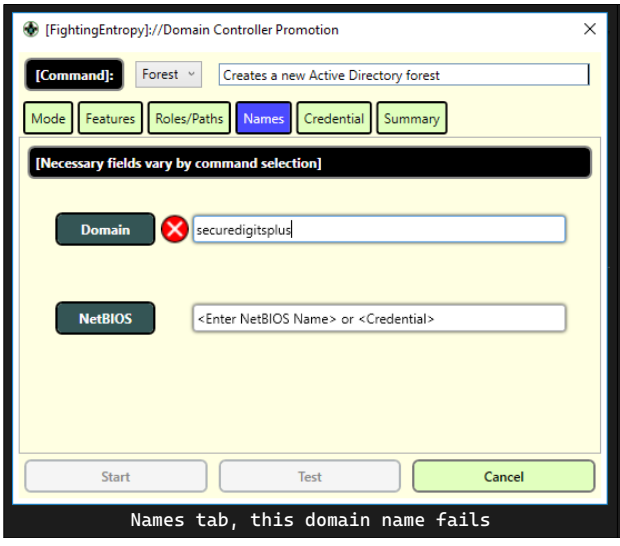
```
   72 SecurityDescriptor RS
   73 SecurityDescriptor RU
   74 SecurityDescriptor SA
   75 SecurityDescriptor SI
   76 SecurityDescriptor SO
   77 SecurityDescriptor SU
   78 SecurityDescriptor SY
   79 SecurityDescriptor WD

PS Prompt:\>
```

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

Name    : Execution
Summary : {<FEDCPromo.ProfileSlotItem>, <FEDCPromo.ProfileSlotItem>, <FEDCPromo.ProfilePasswordItem>...}
Feature : {}
Result  : {}
Output  : {}

PS Prompt:\>
```

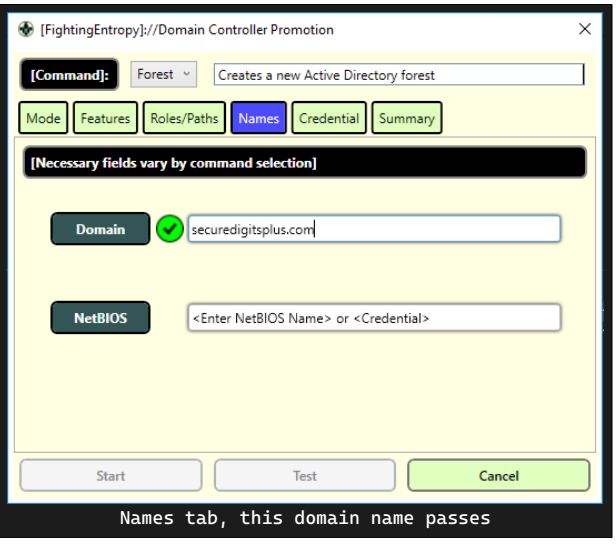
```
PS Prompt:\> $Ctrl.Execution.Summary | Format-Table

Index Name              Type    Property IsEnabled Value               Check Reason
----- ----              ----    -------- --------- -----               ----- ------
    5 DomainName        TextBox Text             1 securedigitsplus.com     1 [+] Passed
    6 DomainNetBIOSName TextBox Text             1 SECURED                  1 [+] Passed
    0 Password                                     <Notapassword!>          1 [+] Passed
    1 Confirm                                      <Notapassword!>          1 [+] Passed

PS Prompt:\>
```

But- there's some information missing here, isn't there...?

Yeah.

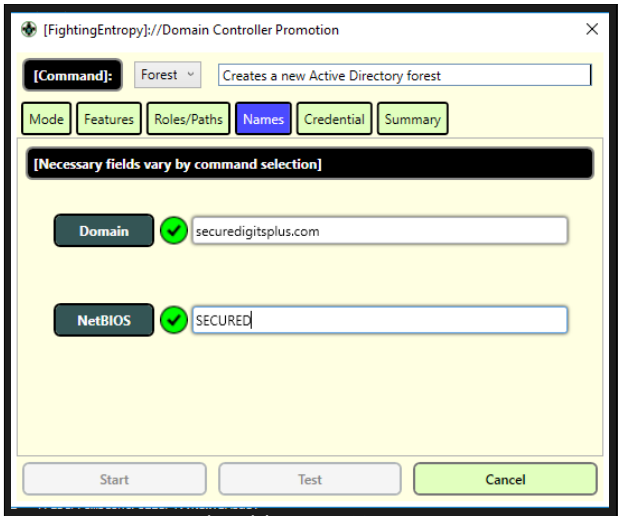


Names tab, this domain name fails

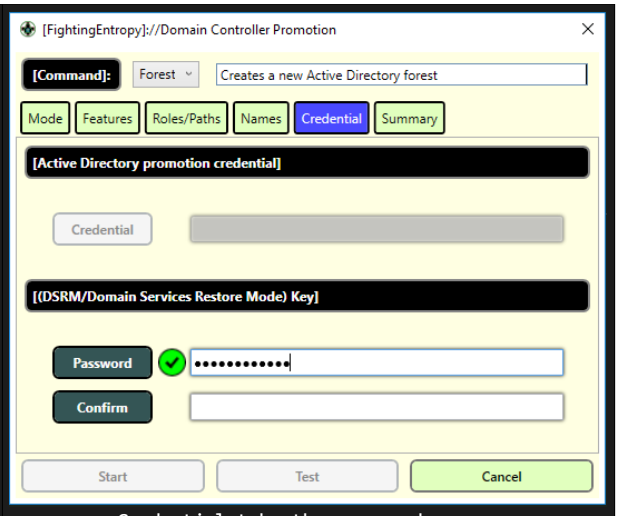


Names tab, this domain name passes

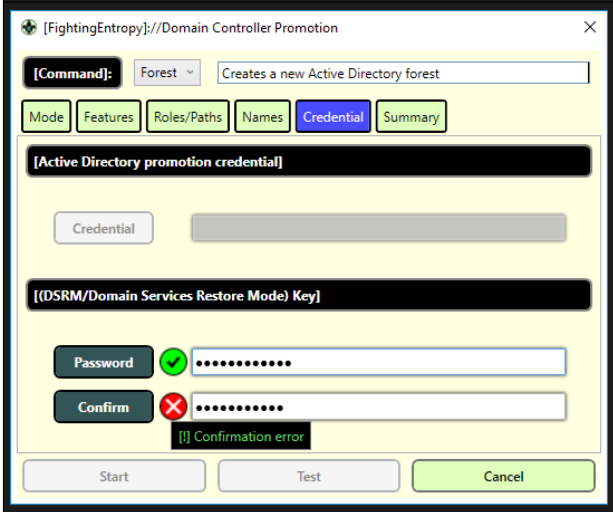
On the next page are the remaining screenshots...



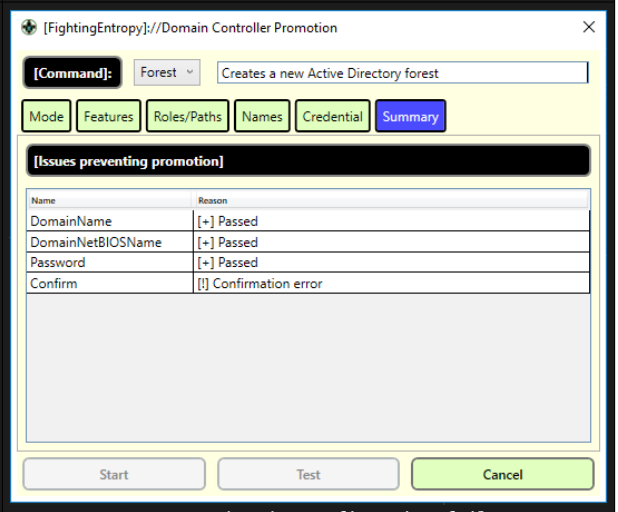
Names tab, this NetBIOS name passes



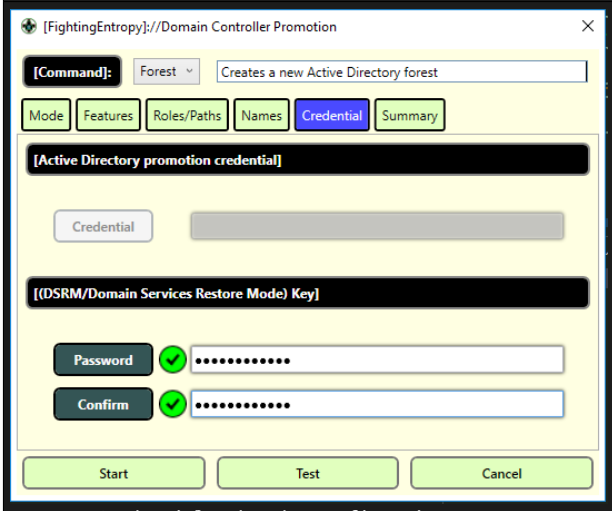
Credential tab, the password passes



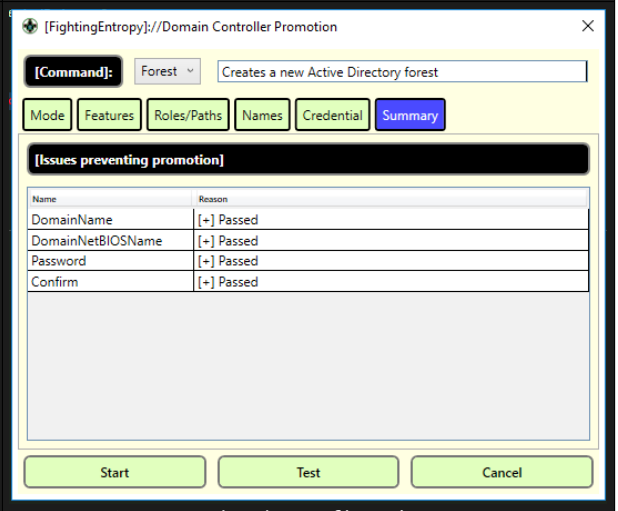
Credential tab, the confirmation fails



Summary tab, the confirmation fails



Credential tab, the confirmation passes



Summary tab, the confirmation passes

_____/ Output

Conclusion

So, there are a number of things that I haven't covered in greater detail in this particular document.

First of all, the fact of the matter is that this function still is not quite complete, as I have to make certain that the changes I've made to the module itself, as well as the other functions... that they allow the function New-FEInfrastructure to contain the modifications and alterations necessary to fulfill the same criteria as the last version.

Not to mention, uh- I'm having to go back and cover other aspects that I haven't fully fleshed out.

I could imagine that when the experts who made Windows version 1 through Windows 95, and onward...|
...made Windows version 1 and onward...
...that they would occasionally gather around in a room, or around a round table...
...and literally mull over what they should name certain switches, properties, objects, classes...
...and if the ideas that some of them had didn't receive a unanimous “Hell yeah, dude”...?

...then somebody was gonna give them the eyebrow movement of doom.

The eyebrow movement of doom, is when someone hears something that sounds really ridiculous, and so they raise at least (1) eyebrow, in a way to where it shows that they do not agree with the thing that caused them to raise at least (1) eyebrow.

Sometimes, they'd have a double eyebrow movement of doom, and that meant that they flat out, could not disagree more, with the idea that they just heard.

I could imagine that some people reading this might say “What about the crossing of the arms...?”
Sure. But I mean, that requires more effort than the eyebrow movement of doom.
Or even a double eyebrow movement of doom.

At that point, the crossing of the arms just adds another layer of disagreement that was already obvious...

Look, nobody wants to experience the intensity behind the concerned look on somebody's face, who couldn't disagree with you even more, by the casual expression of (1-2) eyebrow movements of doom + a crossing of the arms.

And that's just the way life is... people wandering around occasionally having to casually raise (1-2) eyebrows whenever they hear or see something incredibly questionable or ridiculous.

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