```
 ____    ______________________________________________________________________________________________________
//¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
\\__//¯¯¯ Search-WirelessNetwork [~] (11-21-2022)                                                      ___//¯¯\\
 ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
     ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
```

Introduction

In this document, I'm going to discuss the various design implementations I've been making, in reference to the utility listed above.

Over the last few weeks, I have been shoring up the module, [FightingEntropy(π)].
I've made MANY modifications since I made this video about a month ago...

10/28/22	2022_1028-([FightingEntropy(π)][2022.10.1])	https://youtu.be/S7k4lZdPE-I

As well as this document...

10/31/22	https://github.com/mcc85s/FightingEntropy/blob/main/Docs/2022_1031-(FightingEntropy).pdf

Truth be told...?
I've been slowly implementing changes, so that the module can have it's first public release.

When Microsoft, or other technology giants make changes to their codebases, they have to consider making changes that don't break the other components underneath some big, giant (product/software package) that many (companies/agencies) all around the world, use. For instance:

Log4J	https://en.wikipedia.org/wiki/Log4j
CISA - Log4J Vulnerability	https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance

As for Log4J, I'll give the reader a quick summary of what THAT is. It is a hidden nightmare.
Effectively, sometimes a (package/program) has a dependency that relies upon on a certain codebase.
Sometimes, those codebases are rather OBSCURE and extremely difficult to keep track of.

In reality, there really is no way for people to be able to see THAT far ahead into the future, and to build something that never breaks from the ground up. That's just, a REALLY tall order to be able to pull that off.

That said...? That doesn't mean that there aren't better design philosophies where there is less dependence on these obscure packages and et cetera.

In this particular document, I'm going to cover the CURRENT STATE of the function as well as the input/output.

Introduction

Script

```
<#
.SYNOPSIS

    Allows for the management of wireless adapters, profiles, and networks from the
    (CLI/command line interface), or (GUI/graphical user interface) via PowerShell.

.DESCRIPTION

    After seeing various scripts on the internet related to [wireless network]
    management, I decided to build a hybrid (CLI/GUI) utility that is able to:

    1) provide a detailed enumeration of [wireless network] adapter(s),
    2) (create/manage/view) [wireless network] profile(s),
    3) scan for [wireless network](s),
    4) (connect to/disconnect from) [wireless network](s) as well as password prompts,
    5) either from the (command line interface/CLI) (mode 0),
    6) or from the (graphical user interface/GUI) (mode 1).

    [jcwalker] wrote most of the C# code that accesses the wlanapi.dll file, through a manner that
    is similar to P/Invoke. I have implemented his original code (largely verbatim), by modifying
    the signature with the correct namespace and class name, and it is written to:
```

```
    ------------------------------------------------------------
    | $Module   = Get-FEModule -Mode 1                         |
    | $FilePath = $Module.File("Control","Wifi.cs").Fullname |
    ------------------------------------------------------------

    The file and its definitions, are accessible via Add-Type $FilePath, which is implemented by
    the control class, [WirelessController], at the tail end of this function.

    The [WirelessController] class actually has many methods that I've written, as well as some of
    the functions that jcwalker ALSO wrote, to interact with the classes in the type definition.

    Prior to working with the files that interact with wlanapi.dll, I was using I/O from netsh
    and parsing all of it, in order to write the prior version of this function. Wasn't really
    doing a good job at being wicked consistent, because then each output needed to be scoped
    and prepared for, and that's when I realized "This method sorta sucks..."

    Netsh itself is an incredibly (powerful/useful) tool, but- it is complicated.

    As for this particular (function/class), I've added a lot of other components to what
    jcwalker did with his module referenced below, in his Github project.

    The FUNCTIONS written in that module, made more sense to convert to METHODS of a
    (base/factory) class... especially if the (CLI/GUI) controls are doing the same exact thing
    in the code behind.

    So, it felt like it'd be a pretty good idea to capitalize on, as it would be incredibly
    useful in a PXE environment (pretty sure that System Center Configuration Manager has something
    that does that, already).

    As for the PXE environment, that is sorta what I wanted to expand upon and build new features
    for, when I originally recorded this demonstration on 01/25/2019:
    ------------------------------------------------------------------------
    | 2019_0125-(Computer Answers - MDT) | https://youtu.be/5Cyp3pqIMRs |
    ------------------------------------------------------------------------

    THAT particular video showcases how I was originally working with Oracle VirtualBox, to use the
    Microsoft Deployment Toolkit to deploy the Windows operating system installations over a network
    that I (managed/configured) between (2017-2019), at:
    ------------------------------------------------------
    | Computer Answers | 1602 Route 9, Clifton Park, NY |
    ------------------------------------------------------

    Since then, the idea of capitalizing on (PowerShell + Veridian) was my main focus.
    However, this utility is also pretty important.

.LINK

    --------------
    | References |
    --------------

    [ALL_FRONT_RANDOM/Reddit]
    https://www.reddit.com/r/sysadmin/comments/9az53e/need_help_controlling_wifi

    [jcwalker/Github]
    https://github.com/jcwalker/WiFiProfileManagement

    [Wireless Network Scanner (former version)]
    https://youtu.be/35EabWfh8dQ

.NOTES

    ____                                                                                                  ________
   //--\\_____________________________________________________________________________________________________//--\\__//
   \\__//-----------------------------------------------------------------------------------------------------\\__//--\\
   //--\\__[ [FightingEntropy()][2022.11.0] ]_____________________________________________________________//--\\__//
   \\__//-------------------------------------------------------------------------------------------    ---\\
   //---                                                                                                    //
   \\                                                                                                     \\
   //        FileName   : Search-WirelessNetwork.ps1                                                      //
   \\        Solution   : [FightingEntropy()][2022.11.0]                                                  \\
   //        Purpose    : For scanning wireless networks (eventually for use in a PXE environment).       //
   \\        Author     : Michael C. Cook Sr.                                                             \\
   //        Contact    : @mcc85s                                                                         //
   \\        Primary    : @mcc85s                                                                         \\
   //        Created    : 2022-10-10                                                                      //
   \\        Modified   : 2022-11-20                                                                      \\
   //        Demo       : https://youtu.be/35EabWfh8dQ                                                    //
```

```
   \\        Version    : 0.0.0 - () - Finalized functional version 1.                                                  \\
   //        TODO       : N/A                                                                                         ___//
   \\___                                                                                                          ___//--\\
   //--\\_____________________________________________________________________________________________________//---  ___//
   \\__//-----------------------------------------------------------------------------------------------------\\__//---
    ---\\__[ 11-20-2022 17:57:33    ]_____________________________________________________________________//---
        ---------------------------------------------------------------------------------------------------
```

Type	Name	Description
Class	WirelessNetworkXaml	Main GUI/Xaml string
Class	PassphraseXaml	Passphrase GUI/Xaml
Class	XamlProperty	Used to index/catalog the Xaml control objects
Class	XamlWindow	Constructs the XamlWindow object
Enum	PhysicalType	Enum for an SSID's physical network type
Class	PhysicalSlot	Object for an SSID's physical network type
Class	PhysicalList	A list of potential SSID physical network types
Enum	AuthenticationType	Enum for an SSID's authentication type
Class	AuthenticationSlot	Object for an SSID's authentication type
Class	AuthenticationList	A list of potential SSID's authentication types
Enum	EncryptionType	Enum for an SSID's encryption type
Class	EncryptionSlot	Object for an SSID's encryption type
Class	EncryptionList	A list of potential SSID's encryption types
Class	Ssid	Representation of each SSID collected by the wireless radio(s)
Class	SsidSubcontroller	Subcontroller for Ssid information injection
Class	ConnectionModeResolver	(Struct), better than a hashtable
Enum	ConnectionModeType	Enumerates the connection mode in profile object
Class	ConnectionModeSlot	Object for a profile's range of available options
Class	ConnectionModeList	A list for connection modes
Class	WiFiProfile	Shorthand version of the profile object
Class	WiFiProfileExtensionProperty	Compartmentalizes the properties for the WiFiProfile in the Xaml
Class	WifiProfileExtension	Expanded version of the profile object, includes adapter/interface
Class	WifiProfileList	Object that handles a list of profiles on a given adapter
Class	WifiInterface	This deals exclusively with Wireless network adapters
Class	WifiInterfaceNetsh	This retrieves SOME information from netsh
Class	WifiProfileSubcontroller	Handles the profile objects
Class	RtMethod	Specifically for selecting/filtering a Runtime IAsyncTask
Class	WirelessSubcontroller	Specifically for handling (adapters/interfaces), or networks
Class	WirelessController	Controller class for the function, this encapsulates the XAML/GUI

```
.Example
#>
```

Now, I'm going to provide all of the individual classes, one-by-one, each in it's own section.
The point of doing this, is so that (OTHER PEOPLE/I) can review this document at some later point in time, and look for any specific key signatures or methologies that I used in this instance of the function, in order to reconstitute what worked here and now, if something BREAKS in the future.

But also, to (exhibit/explain) what each of these various classes are for.
The truth is, that's a LOT of classes, and there are even more in jcwalker's C# code in the file that is deployed by the upcoming version of [FightingEntropy(π)].

```
                                                                                                                  ________/
\_________________________________________________________________________________________________________________/ Script
  WirelessNetworkXaml /-------------------------------------------------------------------------------------------------\
/---------------------
```

This particular class is not going to show in this document without text wrapping. That's what happens when you use here [String]'s... This particular class isn't showing a herestring, but the intended result of this class, is to produce one.

```
    Class WirelessNetworkXaml
    {
        Static [String] $Content = @(
        '<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="[FightingEntropy]://Wireless Network Scanner" Width="800"
Height="650" HorizontalAlignment="Center" Topmost="True" ResizeMode="CanResizeWithGrip" Icon="C:\ProgramData\Secure
Digits Plus LLC\FightingEntropy\2022.11.0\Graphics\icon.ico" FontFamily="Consolas"
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.Triggers>',
'                <Trigger Property="AlternationIndex" Value="0">',
'                    <Setter Property="Background" Value="White"/>',
'                </Trigger>',
'                <Trigger Property="AlternationIndex" Value="1">',
'                    <Setter Property="Background" Value="#FFC5E5EC"/>',
'                </Trigger>',
'                <Trigger Property="AlternationIndex" Value="2">',
'                    <Setter Property="Background" Value="#FFFDE1DC"/>',
'                </Trigger>',
```

```
        '            </Style.Triggers>',
        '        </Style>',
        '        <Style TargetType="DataGridColumnHeader">',
        '            <Setter Property="FontSize"   Value="10"/>',
        '            <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="FontSize" Value="12"/>',
        '            <Setter Property="Background" Value="Black"/>',
        '            <Setter Property="Foreground" Value="White"/>',
        '            <Setter Property="BorderBrush" Value="Gray"/>',
        '            <Setter Property="BorderThickness" Value="2"/>',
        '            <Setter Property="VerticalContentAlignment" Value="Center"/>',
        '            <Style.Resources>',
        '                <Style TargetType="Border">',
        '                    <Setter Property="CornerRadius" Value="5"/>',
        '                </Style>',
        '            </Style.Resources>',
        '        </Style>',
        '    </Window.Resources>',
        '    <Grid>',
        '        <Grid.Background>',
        '            <ImageBrush Stretch="Fill" ImageSource="C:\ProgramData\Secure Digits Plus
LLC\FightingEntropy\2022.11.0\Graphics\background.jpg"/>',
        '        </Grid.Background>',
        '        <Grid Margin="5">',
        '            <Grid.RowDefinitions>',
        '                <RowDefinition Height="80"/>',
        '                <RowDefinition Height="40"/>',
        '                <RowDefinition Height="40"/>',
        '                <RowDefinition Height="*"/>',
        '            </Grid.RowDefinitions>',
        '            <DataGrid Grid.Row="0" Name="Adapter">',
        '                <DataGrid.Columns>',
        '                    <DataGridTextColumn Header="#"           Width="25"  Binding="{Binding Index}"/>',
        '                    <DataGridTextColumn Header="Name"        Width="200" Binding="{Binding Name}"/>',
        '                    <DataGridTextColumn Header="Description" Width="*"   Binding="{Binding Description}"/>',
        '                </DataGrid.Columns>',
        '            </DataGrid>',
        '            <Grid Grid.Row="1">',
        '                <Grid.ColumnDefinitions>',
        '                    <ColumnDefinition Width="90"/>',
        '                    <ColumnDefinition Width="50"/>',
        '                    <ColumnDefinition Width="90"/>',
        '                    <ColumnDefinition Width="165"/>',
        '                    <ColumnDefinition Width="85"/>',
        '                    <ColumnDefinition Width="*"/>',
        '                </Grid.ColumnDefinitions>',
        '                <Label   Grid.Column="0" Content="[Index]:"/>',
        '                <TextBox Grid.Column="1" Name="Index" IsReadOnly="True"/>',
        '                <Label   Grid.Column="2" Content="[Status]:"/>',
        '                <TextBox Grid.Column="3" Name="Status" IsReadOnly="True"/>',
        '                <Label   Grid.Column="4" Content="[Mac]:"/>',
        '                <TextBox Grid.Column="5" Name="MacAddress" IsReadOnly="True"/>',
        '            </Grid>',
        '            <Grid Grid.Row="2">',
        '                <Grid.ColumnDefinitions>',
        '                    <ColumnDefinition Width="85"/>',
        '                    <ColumnDefinition Width="310"/>',
        '                    <ColumnDefinition Width="85"/>',
        '                    <ColumnDefinition Width="*"/>',
        '                </Grid.ColumnDefinitions>',
        '                <Grid.RowDefinitions>',
        '                    <RowDefinition Height="*"/>',
        '                </Grid.RowDefinitions>',
        '                <Label   Grid.Column="0" Content="[SSID]:"/>',
        '                <TextBox Grid.Column="1" Name="SSID" IsReadOnly="True"/>',
        '                <Label   Grid.Column="2" Content="[BSSID]:"/>',
```

```
        '                <TextBox Grid.Column="3" Name="BSSID" IsReadOnly="True"/>',
        '            </Grid>',
        '            <TabControl Grid.Row="3">',
        '                <TabItem Header="Profile">',
        '                    <Grid>',
        '                        <Grid.RowDefinitions>',
        '                            <RowDefinition Height="*"/>',
        '                            <RowDefinition Height="*"/>',
        '                        </Grid.RowDefinitions>',
        '                        <DataGrid Grid.Row="0" RowHeaderWidth="0" Name="Profile">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn     Header="#"          Width="25"   Binding="{Binding
Index}"/>',
        '                                <DataGridTextColumn     Header="Name"       Width="*"    Binding="{Binding
ProfileName}"/>',
        '                                <DataGridTemplateColumn Header="Connection Mode" Width="140">',
        '                                    <DataGridTemplateColumn.CellTemplate>',
        '                                        <DataTemplate>',
        '                                            <ComboBox SelectedIndex="{Binding ConnectionMode.Index}"
 ToolTip="{Binding ConnectionMode.Description}" Margin="0" Padding="2" Height="18" FontSize="10" IsEnabled="False">',
        '                                                <ComboBoxItem Content="Manual"/>',
        '                                                <ComboBoxItem Content="Auto"/>',
        '                                            </ComboBox>',
        '                                        </DataTemplate>',
        '                                    </DataGridTemplateColumn.CellTemplate>',
        '                                </DataGridTemplateColumn>',
        '                                <DataGridTemplateColumn Header="Authentication" Width="140">',
        '                                    <DataGridTemplateColumn.CellTemplate>',
        '                                        <DataTemplate>',
        '                                            <ComboBox SelectedIndex="{Binding Authentication.Index}" Margin="0"
Padding="2" Height="18" FontSize="10" IsEnabled="False">',
        '                                                <ComboBoxItem Content="None"/>',
        '                                                <ComboBoxItem Content="Unknown"/>',
        '                                                <ComboBoxItem Content="Open80211"/>',
        '                                                <ComboBoxItem Content="SharedKey80211"/>',
        '                                                <ComboBoxItem Content="Wpa"/>',
        '                                                <ComboBoxItem Content="WpaPsk"/>',
        '                                                <ComboBoxItem Content="WpaNone"/>',
        '                                                <ComboBoxItem Content="Rsna"/>',
        '                                                <ComboBoxItem Content="RsnaPsk"/>',
        '                                                <ComboBoxItem Content="Ihv"/>',
        '                                                <ComboBoxItem Content="Wpa3Enterprise192Bits"/>',
        '                                                <ComboBoxItem Content="Wpa3Sae"/>',
        '                                                <ComboBoxItem Content="Owe"/>',
        '                                                <ComboBoxItem Content="Wpa3Enterprise"/>',
        '                                            </ComboBox>',
        '                                        </DataTemplate>',
        '                                    </DataGridTemplateColumn.CellTemplate>',
        '                                </DataGridTemplateColumn>',
        '                                <DataGridTemplateColumn Header="Encryption" Width="140">',
        '                                    <DataGridTemplateColumn.CellTemplate>',
        '                                        <DataTemplate>',
        '                                            <ComboBox SelectedIndex="{Binding Encryption.Index}" Margin="0"
Padding="2" Height="18" FontSize="10" IsEnabled="False">',
        '                                                <ComboBoxItem Content="None"/>',
        '                                                <ComboBoxItem Content="Unknown"/>',
        '                                                <ComboBoxItem Content="Wep"/>',
        '                                                <ComboBoxItem Content="Wep40"/>',
        '                                                <ComboBoxItem Content="Wep104"/>',
        '                                                <ComboBoxItem Content="Tkip"/>',
        '                                                <ComboBoxItem Content="Ccmp"/>',
        '                                                <ComboBoxItem Content="WpaUseGroup"/>',
        '                                                <ComboBoxItem Content="RsnUseGroup"/>',
        '                                                <ComboBoxItem Content="Ihv"/>',
        '                                                <ComboBoxItem Content="Gcmp"/>',
        '                                                <ComboBoxItem Content="Gcmp256"/>',
        '                                            </ComboBox>',
        '                                        </DataTemplate>',
        '                                    </DataGridTemplateColumn.CellTemplate>',
        '                                </DataGridTemplateColumn>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
```

```
        '                        <DataGrid Grid.Row="1" Name="ProfileExtension" ScrollViewer.CanContentScroll="False"
IsEnabled="False">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="Name"  Binding="{Binding Name}"  Width="100"/>',
        '                                <DataGridTextColumn Header="Value" Binding="{Binding Value}" Width="*"/>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                    </Grid>',
        '                </TabItem>',
        '                <TabItem Header="Network">',
        '                    <Grid>',
        '                        <Grid.RowDefinitions>',
        '                            <RowDefinition Height="40"/>',
        '                            <RowDefinition Height="*"/>',
        '                            <RowDefinition Height="40"/>',
        '                        </Grid.RowDefinitions>',
        '                        <Grid Grid.Row="0">',
        '                            <Grid.ColumnDefinitions>',
        '                                <ColumnDefinition Width="90"/>',
        '                                <ColumnDefinition Width="120"/>',
        '                                <ColumnDefinition Width="*"/>',
        '                                <ColumnDefinition Width="100"/>',
        '                                <ColumnDefinition Width="120"/>',
        '                            </Grid.ColumnDefinitions>',
        '                            <Label    Grid.Column="0" Content="[Filter]:"/>',
        '                            <ComboBox Grid.Column="1" Name="FilterProperty" SelectedIndex="0">',
        '                                <ComboBoxItem Content="Name"/>',
        '                                <ComboBoxItem Content="Index"/>',
        '                                <ComboBoxItem Content="BSSID"/>',
        '                                <ComboBoxItem Content="Type"/>',
        '                                <ComboBoxItem Content="Encryption"/>',
        '                                <ComboBoxItem Content="Strength"/>',
        '                            </ComboBox>',
        '                            <TextBox Grid.Column="2" Name="FilterText"/>',
        '                            <ProgressBar Grid.Column="3" Margin="10" Name="Progress"/>',
        '                            <Button  Grid.Column="4" Name="Refresh"  Content="(Scan/Refresh)"
 IsEnabled="False"/>',
        '                        </Grid>',
        '                        <DataGrid Grid.Row="1" Grid.Column="0" Name="Network" RowHeaderWidth="0">',
        '                            <DataGrid.Columns>',
        '                                <DataGridTextColumn Header="#"        Width="25"  Binding="{Binding Index}"/>',
        '                                <DataGridTextColumn Header="Name"     Width="240" Binding="{Binding Name}"/>',
        '                                <DataGridTextColumn Header="Bssid"    Width="120" Binding="{Binding Bssid}"/>',
        '                                <DataGridTemplateColumn Header="Phy." Width="40">',
        '                                    <DataGridTemplateColumn.CellTemplate>',
        '                                        <DataTemplate>',
        '                                            <ComboBox SelectedIndex="{Binding Physical.Index}"
ToolTip="{Binding Physical.Description}" Margin="0" Padding="2" Height="18" FontSize="10" IsEnabled="False">',
        '                                                <ComboBoxItem Content="Unknown"/>',
        '                                                <ComboBoxItem Content="Fhss"/>',
        '                                                <ComboBoxItem Content="Dsss"/>',
        '                                                <ComboBoxItem Content="IRBaseband"/>',
        '                                                <ComboBoxItem Content="Ofdm"/>',
        '                                                <ComboBoxItem Content="Hrdsss"/>',
        '                                                <ComboBoxItem Content="Erp"/>',
        '                                                <ComboBoxItem Content="HT"/>',
        '                                                <ComboBoxItem Content="Vht"/>',
        '                                                <ComboBoxItem Content="Dmg"/>',
        '                                                <ComboBoxItem Content="HE"/>',
        '                                            </ComboBox>',
        '                                        </DataTemplate>',
        '                                    </DataGridTemplateColumn.CellTemplate>',
        '                                </DataGridTemplateColumn>',
        '                                <DataGridTextColumn Header="Uptime" Width="120" Binding="{Binding Uptime}"/>',
        '                                <DataGridTemplateColumn Header="Auth." Width="75">',
        '                                    <DataGridTemplateColumn.CellTemplate>',
        '                                        <DataTemplate>',
        '                                            <ComboBox SelectedIndex="{Binding Authentication.Index}"
 ToolTip="{Binding Authentication.Description}" Margin="0" Padding="2" Height="18" FontSize="10" IsEnabled="False">',
        '                                                <ComboBoxItem Content="None"/>',
        '                                                <ComboBoxItem Content="Unknown"/>',
        '                                                <ComboBoxItem Content="Open80211"/>',
```

```
        '                                                <ComboBoxItem Content="SharedKey80211"/>',
        '                                                <ComboBoxItem Content="Wpa"/>',
        '                                                <ComboBoxItem Content="WpaPsk"/>',
        '                                                <ComboBoxItem Content="WpaNone"/>',
        '                                                <ComboBoxItem Content="Rsna"/>',
        '                                                <ComboBoxItem Content="RsnaPsk"/>',
        '                                                <ComboBoxItem Content="Ihv"/>',
        '                                                <ComboBoxItem Content="Wpa3Enterprise192Bits"/>',
        '                                                <ComboBoxItem Content="Wpa3Sae"/>',
        '                                                <ComboBoxItem Content="Owe"/>',
        '                                                <ComboBoxItem Content="Wpa3Enterprise"/>',
        '                                            </ComboBox>',
        '                                        </DataTemplate>',
        '                                    </DataGridTemplateColumn.CellTemplate>',
        '                                </DataGridTemplateColumn>',
        '                                <DataGridTemplateColumn Header="Enc." Width="75">',
        '                                    <DataGridTemplateColumn.CellTemplate>',
        '                                        <DataTemplate>',
        '                                            <ComboBox SelectedIndex="{Binding Encryption.Index}"
ToolTip="{Binding Encryption.Description}" Margin="0" Padding="2" Height="18" FontSize="10" IsEnabled="False">',
        '                                                <ComboBoxItem Content="None"/>',
        '                                                <ComboBoxItem Content="Unknown"/>',
        '                                                <ComboBoxItem Content="Wep"/>',
        '                                                <ComboBoxItem Content="Wep40"/>',
        '                                                <ComboBoxItem Content="Wep104"/>',
        '                                                <ComboBoxItem Content="Tkip"/>',
        '                                                <ComboBoxItem Content="Ccmp"/>',
        '                                                <ComboBoxItem Content="WpaUseGroup"/>',
        '                                                <ComboBoxItem Content="RsnUseGroup"/>',
        '                                                <ComboBoxItem Content="Ihv"/>',
        '                                                <ComboBoxItem Content="Gcmp"/>',
        '                                                <ComboBoxItem Content="Gcmp256"/>',
        '                                            </ComboBox>',
        '                                        </DataTemplate>',
        '                                    </DataGridTemplateColumn.CellTemplate>',
        '                                </DataGridTemplateColumn>',
        '                                <DataGridTemplateColumn Header="Str." Width="40">',
        '                                    <DataGridTemplateColumn.CellTemplate>',
        '                                        <DataTemplate>',
        '                                            <ComboBox SelectedIndex="{Binding Strength}" Margin="0" Padding="2"
Height="18" FontSize="10" IsEnabled="False">',
        '                                                <ComboBoxItem Content="0"/>',
        '                                                <ComboBoxItem Content="1"/>',
        '                                                <ComboBoxItem Content="2"/>',
        '                                                <ComboBoxItem Content="3"/>',
        '                                                <ComboBoxItem Content="4"/>',
        '                                                <ComboBoxItem Content="5"/>',
        '                                            </ComboBox>',
        '                                        </DataTemplate>',
        '                                    </DataGridTemplateColumn.CellTemplate>',
        '                                </DataGridTemplateColumn>',
        '                            </DataGrid.Columns>',
        '                        </DataGrid>',
        '                        <Grid Grid.Row="2">',
        '                            <Grid.ColumnDefinitions>',
        '                                <ColumnDefinition Width="*"/>',
        '                                <ColumnDefinition Width="*"/>',
        '                                <ColumnDefinition Width="*"/>',
        '                            </Grid.ColumnDefinitions>',
        '                            <Button Grid.Column="0" Name="Connect"    Content="Connect"
 IsEnabled="False"/>',
        '                            <Button Grid.Column="1" Name="Disconnect" Content="Disconnect"
IsEnabled="False"/>',
        '                            <Button Grid.Column="2" Name="Cancel"     Content="Cancel"/>',
        '                        </Grid>',
        '                    </Grid>',
        '                </TabItem>',
        '            </TabControl>',
        '        </Grid>',
        '    </Grid>',
        '</Window>' -join "`n")
    }
```

```
                                                                                                                  _____________________/
\_____________________________________________________________________________________________________________/ WirelessNetworkXaml
  PassphraseXaml /-------------------------------------------------------------------------------------------------------------------\
/----------------
```

    This is another chunk of Xaml that is meant specifically to ask for the passphrase of a target wireless network
    that just so happens to ask for a password. Some target network types that will be described later on, they do
    not actually use a PASSPHRASE, they may use a CERTIFICATE or something to that effect.

```
    Class PassphraseXaml
    {
        Static [String] $Content = @(
        '<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="[FightingEntropy]://Enter Passphrase" Width="400"
Height="160" HorizontalAlignment="Center" Topmost="True" ResizeMode="CanResizeWithGrip" Icon="C:\ProgramData\Secure
Digits Plus LLC\FightingEntropy\2022.11.0\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 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"/>',
        '            <Setter Property="PasswordChar" Value="*"/> ',
        '        </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>',
        '    </Window.Resources>',
        '    <Grid>',
        '        <Grid.Background>',
        '            <ImageBrush Stretch="Fill" ImageSource="C:\ProgramData\Secure Digits Plus
LLC\FightingEntropy\2022.11.0\Graphics\background.jpg"/>',
        '        </Grid.Background>',
        '        <GroupBox>',
        '            <Grid>',
        '                <Grid.RowDefinitions>',
        '                    <RowDefinition Height="*"/>',
        '                    <RowDefinition Height="*"/>',
        '                </Grid.RowDefinitions>',
        '                <PasswordBox Grid.Row="0" Name="Passphrase"/>',
        '                <Grid Grid.Row="1">',
        '                    <Grid.ColumnDefinitions>',
        '                            <ColumnDefinition Width="*"/>',
        '                            <ColumnDefinition Width="*"/>',
        '                        </Grid.ColumnDefinitions>',
        '                    <Button Grid.Row="1" Grid.Column="0" Name="Connect" Content="Continue"/>',
        '                    <Button Grid.Row="1" Grid.Column="2" Name="Cancel"  Content="Cancel"/>',
        '                </Grid>',
        '            </Grid>',
        '        </GroupBox>',
        '    </Grid>',
        '</Window>' -join "`n")
    }
```

XamlProperty / PassphraseXaml

This class is specifically meant for (INDEXING/CATEGORIZING) the objects that [PowerShell] and the Windows Presentation Framework.dll objects instantiate, when converting the above chunks of Xaml, into tangible objects on the screen.

```
    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
        }
    }
```

```
                                                                                                                                                  _______________/
\_____________________________________________________________________________________________________________________________________________/ XamlProperty
  XamlWindow /-------------------------------------------------------------------------------------------------------------------------------------\
/------------
```

This is a class that I have been shaping and molding over the course of the last ~(4) years or so once I started working with Xaml and [PowerShell], and it uses a combination of techniques that I figured out how to use after watching some of Tobias Weltner's and Damien Van Robaeys' (demonstrations/scripts).

```
    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)
        }
        Invoke()
        {
            Try
            {
                $This.IO.Dispatcher.InvokeAsync({ $This.IO.ShowDialog() }).Wait()
            }
            Catch
            {
                $This.Exception = $PSItem
            }
```

```
        }
    }
```

_____/ XamlWindow
PhysicalType /_____\

/_____

This is an Enum type meant for categorizing the possible physical radio network types.
Each of these strings are a lot like VGA, XVGA, SVGA, et cetera.
HT means High Throughput/802.11n. VHT means Very High Throughput/802.11ac.

```
Enum PhysicalType
{
    Unknown
    Fhss
    Dsss
    IRBaseband
    Ofdm
    Hrdsss
    Erp
    HT
    Vht
    Dmg
    HE
}
```

_____/ PhysicalType
PhysicalSlot /_____\

/_____

This particular class is going to be repeated to some degree a fair amount, but it is meant to specifically
provide in individual slot for the above Enum class to build itself into a list of these objects with their
corresponding numerical INDEX, the TYPE (the strings in the above class), and the DESCRIPTION (what it is).

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

_____/ PhysicalSlot
PhysicalList /_____\

/_____

This is effectively, the list that would be comprised of the above (2) classes, and...
It is what allows the Xaml to be able to CLEANLY select the correct class type from the radio scanner classes.

However, this information is ALSO able to be obtained using the CLI method of this same exact function.

```
Class PhysicalList
{
```

```
    [Object] $Output
    PhysicalList()
    {
        $This.Output = @( )
        [System.Enum]::GetNames([PhysicalType]) | % { $This.Add($_) }
    }
    Add([String]$Name)
    {
        $Item             = [PhysicalSlot]::New($Name)
        $Item.Description = Switch ($Name)
        {
            Unknown     { "Unspecified physical type"                      }
            Fhss        { "(FHSS/Frequency-Hopping Spread-Spectrum)"       }
            Dsss        { "(DSSS/Direct Sequence Spread-Spectrum)"         }
            IRBaseband  { "(IR/Infrared baseband)"                         }
            Ofdm        { "(OFDM/Orthogonal Frequency Division Multiplex)" }
            Hrdsss      { "(HRDSSS/High-rated DSSS)"                       }
            Erp         { "(ERP/Extended Rate)"                            }
            HT          { "(HT/High Throughput [802.11n])"                 }
            Vht         { "(VHT/Very High Throughput [802.11ac])"          }
            Dmg         { "(DMG/Directional Multi-Gigabit [802.11ad])"     }
            HE          { "(HEW/High-Efficiency Wireless [802.11ax])"      }
        }
        $This.Output += $Item
    }
    [Object] Get([String]$Type)
    {
        Return $This.Output[[UInt32][PhysicalType]::$Type]
    }
}
```

```
                                                                                              ______________/
\____________________________________________________________________________________________/ PhysicalList
  AuthenticationType /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

The numbers do not actually NEED to be there, but since they are, it just illustrates that having an enum type like this is specifically meant to iterate through a list of (strings/names), and a specific (string/name) will return a specific integer value.

```
Enum AuthenticationType
{
    None                    = 0
    Unknown                 = 1
    Open80211               = 2
    SharedKey80211          = 3
    Wpa                     = 4
    WpaPsk                  = 5
    WpaNone                 = 6
    Rsna                    = 7
    RsnaPsk                 = 8
    Ihv                     = 9
    Wpa3                    = 10
    Wpa3Sae                 = 11
    Owe                     = 12
    Wpa3Enterprise          = 13
}
```

```
                                                                                        ____________________/
\______________________________________________________________________________________/ AuthenticationType
  AuthenticationSlot /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Some of the responses from the radio will actually come back as Wpa3Enterprise192Bits, which is the same thing as Wpa3. However, I believe that as (IEEE/Institute of Electrical and Electronics Engineers) develops new protocols, radio types, standards, and whatnot...?

They're shooting for a high level of excellence in their engineering.

So, having such a high level of excellence is going to cause them to sit around at a table sometimes, and say:

/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯\

IEEE Guy[1]: Hey, maybe WPA3 and WPA3ENTERPRISE192BITS should be separate...
IEEE Guy[2]: Maybe...
IEEE Guy[1]: We wouldn't want to use redundant terms or anything like that if we make amendments later on, right?
IEEE Guy[2]: No way, IEEE Guy[1]...
IEEE Guy[1]: Alright.

__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/

```
Class AuthenticationSlot
{
    [UInt32]         $Index
    [String]          $Type
    [String[]] $Description
    AuthenticationSlot([String]$xType)
    {
        If ($xType -eq "Wpa3Enterprise192Bits")
        {
            $xType = "Wpa3"
        }
        $This.Type        = [AuthenticationType]::$xType
        $This.Index       = [UInt32][AuthenticationType]::$xType
    }
    [String] ToString()
    {
        Return $This.Type
    }
}
```

```
                                                                                                    ____________________/
\__________________________________________________________________________________________________/ AuthenticationSlot
  AuthenticationList /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Here is the second instance of the “List” object being repeated, but for the `[AuthenticationType]`.

```
Class AuthenticationList
{
    [Object] $Output
    AuthenticationList()
    {
        $This.Output  = @( )
        [System.Enum]::GetNames([AuthenticationType]) | % { $This.Add($_) }
    }
    Add([String]$Type)
    {
        $Item              = [AuthenticationSlot]::New($_)
        $Item.Description = Switch -Regex ($Item.Type)
        {
            ^None$
            {
                "No authentication enabled."
            }
            ^Unknown$
            {
                "Authentication method unknown."
            }
            ^Open80211$
            {
                "Open authentication over 802.11 wireless.",
                "Devices are authenticated and can connect to an access point.",
                "Communication w/ network requires matching (WEP/Wired Equivalent Privacy) key."
            }
            ^SharedKey80211$
            {
```

```
                    "Specifies an IEEE 802.11 Shared Key authentication algorithm.",
                    "Requires pre-shared (WEP/Wired Equivalent Privacy) key for 802.11 authentication."
                }
                ^Wpa$
                {
                    "Specifies a (WPA/Wi-Fi Protected Access) algorithm.",
                    "IEEE 802.1X port authorization is performed by the supplicant, authenticator, and authentication
server.",
                    "Cipher keys are dynamically derived through the authentication process."
                }
                ^WpaPsk$
                {
                    "Specifies a (WPA/Wi-Fi Protected Access) algorithm that uses (PSK/pre-shared key).",
                    "IEEE 802.1X port authorization is performed by the supplicant and authenticator.",
                    "Cipher keys are dynamically derived through a PSK that is used on both the supplicant and
authenticator."
                }
                ^WpaNone$
                {
                    "Wi-Fi Protected Access."
                }
                ^Rsna$
                {
                    "Specifies an IEEE 802.11i (RSNA/Robust Security Network Association) algorithm.",
                    "IEEE 802.1X port authorization is performed by the supplicant, authenticator, and authentication
server.",
                    "Cipher keys are dynamically derived through the auth. process."
                }
                ^RsnaPsk$
                {
                    "Specifies an IEEE 802.11i RSNA algorithm that uses (PSK/pre-shared key).",
                    "IEEE 802.1X port authorization is performed by the supplicant and authenticator.",
                    "Cipher keys are dynamically derived through a PSK that is used on both the supplicant and
authenticator."
                }
                ^Ihv$
                {
                    "Specifies an authentication type defined by an (IHV/Independent Hardware Vendor)."
                }
                "(^Wpa3$|^Wpa3Enterprise192Bits$)"
                {
                    "Specifies a 192-bit encryption mode for (WPA3-Enterprise/Wi-Fi Protected Access 3 Enterprise)
networks."
                }
                ^Wpa3Sae$
                {
                    "Specifies (WPA3 SAE/Wi-Fi Protected Access 3 Simultaneous Authentication of Equals) algorithm.",
                    "WPA3 SAE is the consumer version of WPA3. SAE is a secure key establishment protocol between
devices;",
                    "SAE provides: synchronous authentication, and stronger protections for users against",
                    "password-guessing attempts by third parties."
                }
                ^Owe$
                {
                    "Specifies an (OWE/Opportunistic Wireless Encryption) algorithm.",
                    "OWE provides opportunistic encryption over 802.11 wireless networks.",
                    "Cipher keys are dynamically derived through a (DH/Diffie-Hellman) key exchange-",
                    "Enabling data protection without authentication."
                }
                ^Wpa3Enterprise$
                {
                    "Specifies a (WPA3-Enterprise/Wi-Fi Protected Access 3 Enterprise) algorithm.",
                    "WPA3-Enterprise uses IEEE 802.1X in a similar way as (RSNA/Robust Security Network Association).",
                    "However, it provides increased security through the use of mandatory certificate validation and
protected management frames."
                }
            }
            $This.Output += $Item
        }
        [Object] Get([String]$Type)
        {
            Return $This.Output[[UInt32][AuthenticationType]::$Type]
```

```
        }
    }
```

_____/
_____/ AuthenticationList
EncryptionType /_____\
/_____

Enumerating these encryption types does not need the number there at all, as I mentioned with the previous enum.

```
    Enum EncryptionType
    {
        None        = 0
        Unknown     = 1
        Wep         = 2
        Wep40       = 3
        Wep104      = 4
        Tkip        = 5
        Ccmp        = 6
        WpaUseGroup = 7
        RsnUseGroup = 8
        Ihv         = 9
        Gcmp        = 10
        Gcmp256     = 11
    }
```

_____/
_____/ EncryptionType
EncryptionSlot /_____\
/_____

Single object meant for a list of `[EncryptionType]`'s.

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

_____/
_____/ EncryptionSlot
EncryptionList /_____\
/_____

Here is the third instance of the “List” object being repeated, but for the `[EncryptionType]`.

```
    Class EncryptionList
    {
        [Object] $Output
        EncryptionList()
        {
            $This.Output = @( )
            [System.Enum]::GetNames([EncryptionType]) | % { $This.Add($_) }
        }
```

```
        Add([String]$Type)
        {
            $Item             = [EncryptionSlot]::New($Type)
            $Item.Description = Switch ($Item.Type)
            {
                None
                {
                    "No encryption enabled."
                }
                Unknown
                {
                    "Encryption method unknown."
                }
                Wep
                {
                    "Specifies a WEP cipher algorithm with a cipher key of any length."
                }
                Wep40
                {
                    "Specifies an RC4-based (WEP/Wired Equivalent Privacy) algorithm specified in IEEE 802.11-1999.",
                    "This enumerator specifies the WEP cipher algorithm with a 40-bit cipher key."
                }
                Wep104
                {
                    "Specifies a (WEP/Wired Equivalent Privacy) cipher algorithm with a 104-bit cipher key."
                }
                Tkip
                {
                    "Specifies an RC4-based cipher (TKIP/Temporal Key Integrity Protocol) algorithm",
                    "This cipher suite that is based on algorithms defined in WPA + IEEE 802.11i-2004 standards.",
                    "This cipher also uses the (MIC/Message Integrity Code) algorithm for forgery protection."
                }
                Ccmp
                {
                    "Specifies an [IEEE 802.11i-2004 & RFC 3610] AES-CCMP algorithm standard.",
                    "(AES/Advanced Encryption Standard) is the encryption algorithm defined in FIPS PUB 197."
                }
                WpaUseGroup
                {
                    "Specifies a (WPA/Wifi Protected Access) Use Group Key cipher suite.",
                    "For more information about the Use Group Key cipher suite, refer to:",
                    "Clause 7.3.2.25.1 of the IEEE 802.11i-2004 standard."
                }
                RsnUseGroup
                {
                    "Specifies a (RSN/Robust Security Network) Use Group Key cipher suite.",
                    "For more information about the Use Group Key cipher suite, refer to:",
                    "Clause 7.3.2.25.1 of the IEEE 802.11i-2004 standard."
                }
                Ihv
                {
                    "Specifies an encryption type defined by an (IHV/Independent Hardware Vendor)."
                }
                Gcmp
                {
                    "Specifies an [IEEE 802.11-2016] AES-GCMP algorithm w/ 128-bit key.",
                    "(AES/Advanced Encryption Standard) is the encryption algorithm defined in FIPS PUB 197."
                }
                Gcmp256
                {
                    "Specifies an [IEEE 802.11-2016] AES-GCMP algorithm w/ 256-bit key.",
                    "(AES/Advanced Encryption Standard) is the encryption algorithm defined in FIPS PUB 197."
                }
            }
            $This.Output     += $Item
        }
        [Object] Get([String]$Type)
        {
            Return $This.Output[[UInt32][EncryptionType]::$Type]
        }
    }
```

```
                                                                                                                  _________________/
\_____________________________________________________________________________________________________________/ EncryptionList
  Ssid /-------------------------------------------------------------------------------------------------------------------------\
/------
```

This particular class is meant to capture the responses from the radios.
The property [Object]$Object, is actually the object that comes back from the radio.
However, it needs to be (changed/amended) to a rather large degree, in order for the content that it returns, to be CONSUMABLE, (and/or) viewable in a Xaml DataGrid.

In other words, some of the returned information needs the enumeration classes up above (and even some below), in order to logically comparmentalize it in a way that makes sense VISUALLY.

```
Class Ssid
{
    [UInt32]            $Index
    [Object]             $Ssid
    [String]             $Name
    [Object]            $Bssid
    [Object]         $Physical
    [Object]          $Network
    [Object]           $Uptime
    [Object]   $Authentication
    [Object]       $Encryption
    [UInt32]         $Strength
    [String]   $BeaconInterval
    [Double] $ChannelFrequency
    [Bool]       $IsWifiDirect
    Ssid([UInt32]$Index,[Object]$Object)
    {
        $This.Index             = $Index
        $This.Ssid              = $Object
        $This.Name              = If (!$Object.Ssid) { "<Hidden>" } Else { $Object.Ssid }
        $This.Bssid             = $Object.Bssid.ToUpper()
        $This.Network           = $Object.NetworkKind
        $This.Strength          = $Object.SignalBars
        $This.BeaconInterval    = $Object.BeaconInterval
        $This.ChannelFrequency  = $Object.ChannelCenterFrequencyInKilohertz
        $This.IsWiFiDirect      = $Object.IsWiFiDirect
    }
    [String] ToString()
    {
        Return $This.Name
    }
}
```

```
                                                                                                          ______/
\________________________________________________________________________________________________________/ Ssid
  SsidSubcontroller /-------------------------------------------------------------------------------------------\
/-------------------
```

This particular class is specifically meant to prevent each class or representation, from having to multiple or repeat the information that is in the above enumeration types and list classes.

Effectively, what this class is meant to do, is act as a REFERENCE.

```
Class SsidSubcontroller
{
    [Object] $Physical
    [Object] $Authentication
    [Object] $Encryption
    [Object] $Output
    SsidSubcontroller()
    {
        $This.Physical       = [PhysicalList]::New()
        $This.Authentication = [AuthenticationList]::New()
        $This.Encryption     = [EncryptionList]::New()
    }
```

```
        Load([Object]$Ssid)
        {
            $Ssid.Physical       = $This.Physical.Get($Ssid.Ssid.PhyKind)
            $Ssid.Uptime         = $This.GetUptime($Ssid.Ssid.Uptime)
            $Ssid.Authentication = $This.Authentication.Get($Ssid.Ssid.SecuritySettings.NetworkAuthenticationType)
            $Ssid.Encryption     = $This.Encryption.Get($Ssid.Ssid.SecuritySettings.NetworkEncryptionType)
        }
        [String] GetUptime([Object]$Uptime)
        {
            Return @(Switch ($Uptime)
            {
                {$_.Days -gt 0}
                {
                    $Uptime | % { "{0}d {1}h {2}m {3}s" -f $_.Days, $_.Hours, $_.Minutes, $_.Seconds }
                }
                {$_.Days -eq 0 -and $_.Hours -gt 0}
                {
                    $Uptime | % { "{0}h {1}m {2}s" -f $_.Hours, $_.Minutes, $_.Seconds }
                }
                {$_.Hours -eq 0 -and $_.Seconds -gt 0}
                {
                    $Uptime | % { "{0}m {1}s" -f $_.Minutes, $_.Seconds }
                }
            })
        }
    }
```

ConnectionModeResolver

This is essentially the same thing as a struct, I believe.
Classes and structs are almost identical. I really do not know if there's a difference between the two...?
But, the struct type does not exist in [PowerShell], while it DOES in [C#].

This particular class has no main methods or constructors, which means that it's similar to a [Hashtable],
except that it will retain it's order, whereas a [Hashtable] will lose it's key order.
(If I'm wrong about that, feel free to let me know. As far as I know, [Hashtable] objects lose their key order.)

```
    Class ConnectionModeResolver
    {
        [String] $Profile           = "WLAN_CONNECTION_MODE_PROFILE"
        [String] $TemporaryProfile  = "WLAN_CONNECTION_MODE_TEMPORARY_PROFILE"
        [String] $DiscoverySecure   = "WLAN_CONNECTION_MODE_DISCOVERY_SECURE"
        [String] $Auto              = "WLAN_CONNECTION_MODE_AUTO"
        [String] $DiscoveryUnsecure = "WLAN_CONNECTION_MODE_DISCOVERY_UNSECURE"
    }
```

ConnectionModeType

This is specifically meant for the connection mode for the [ProfileXml] stuff.

```
    Enum ConnectionModeType
    {
        Manual = 0
        Auto   = 1
    }
```

ConnectionModeSlot

Single object meant for a list of [ConnectionModeType]'s.

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

_____/
_____/ ConnectionModeSlot
ConnectionModeList /_____\
/_____

Here is the fourth instance of the "List" object being repeated, but for the [ConnectionModeType].

```
Class ConnectionModeList
{
    [Object] $Output
    ConnectionModeList()
    {
        $This.Output = @( )
        [System.Enum]::GetNames([ConnectionModeType]) | % { $This.Add($_) }
    }
    Add([String]$Type)
    {
        $Item             = [ConnectionModeSlot]::New($Type)
        $Item.Description = Switch ($Type)
        {
            Manual
            {
                "Profile for this access point requires manual intervention"
            }
            Auto
            {
                "Profile for this access point is automatic"
            }
        }
        $This.Output     += $Item
    }
    [Object] Get([String]$Type)
    {
        Return $This.Output[[UInt32][ConnectionModeType]::$Type]
    }
}
```

_____/
_____/ ConnectionModeList
WiFiProfile /_____\
/_____

So, part of why jcwalker's code signature is so critical to this utility, is that it accesses "wlanapi.dll" (which is what I believe that netsh ALSO accesses), in order to retrieve information about the wireless adapter(s) in the system. The only other way to access unmanaged code from [PowerShell], is to use [P/Invoke].

With this particular (*.dll), it is able to retrieve the [ProfileXml] objects that are saved in a rather cryptic location in the operating system (it's not THAT cryptic, actually)...

```
Class WiFiProfile
{
    [UInt32] $Index
    [String] $Name
    [String] $Flags
    [Object] $Detail
    WiFiProfile([UInt32]$Index,[Object]$xProfile)
    {
        $This.Index     = $Index
        $This.Name      = $xProfile.strProfileName
        $This.Flags     = $xProfile.ProfileFlags
        $This.Detail    = $Null
    }
}
```

WifiProfileExtensionProperty / WiFiProfile

This particular class is really just about the same thing as a `[PSNoteProperty]` object, and the reason why that isn't used INSTEAD, is mainly because there was a visual anomaly showing up in the DataGrid. It probably does not need to exist, but- I'm going to leave it in anyway.

"It's better to HAVE a tool, and NOT need it... than to NEED a tool, and NOT HAVE it..." -Smart guys

```
Class WifiProfileExtensionProperty
{
    [UInt32] $Index
    [String] $Name
    [String] $Value
    WifiProfileExtensionProperty([UInt32]$Index,[String]$Name,[Object]$Value)
    {
        $This.Index = $Index
        $This.Name  = $Name
        $This.Value = @(Switch ($Value.Count)
        {
            {$_ -eq 0}
            {
                "-"
            }
            {$_ -eq 1}
            {
                $Value
            }
            {$_ -gt 1}
            {
                $Value -join "`n"
            }
        })
    }
}
```

WifiProfileExtension / WifiProfileExtensionProperty

This particular class EXTENDS the properties of the above class `[WiFiProfile]`, as it expands the properties within the property `[Object] $Detail` returned from wlanapi.dll, and then the info from the (adapter/interface) is ALSO applied.

```
Class WifiProfileExtension
{
    [UInt32]               $Index
    Hidden [String]         $Name
```

```
        Hidden [Guid]           $Guid
        Hidden [String]  $Description
        Hidden [UInt32]      $IfIndex
        Hidden [String]       $Status
        Hidden [String]   $MacAddress
        Hidden [String]    $LinkSpeed
        Hidden [String]        $State
        [String]         $ProfileName
        [Object]      $ConnectionMode
        [Object]      $Authentication
        [Object]          $Encryption
        [String]            $Password
        [UInt32]   $ConnectHiddenSSID
        [String]             $EapType
        [String[]]       $ServerNames
        [String]       $TrustedRootCA
        [String]               $Xml
        WifiProfileExtension([Object]$Interface)
        {
            $This.Name              = $Interface.Name
            $This.Guid              = $Interface.Guid
            $This.Description       = $Interface.Description
            $This.IfIndex           = $Interface.IfIndex
            $This.Status            = $Interface.Status
            $This.MacAddress        = $Interface.MacAddress
            $This.LinkSpeed         = $Interface.LinkSpeed
            $This.State             = $Interface.State
        }
        Load([Object]$xProfile)
        {
            $This.ProfileName       = $xProfile.ProfileName
            $This.Password          = $xProfile.Password
            $This.ConnectHiddenSSID = $xProfile.ConnectHiddenSSID
            $This.EapType           = $xProfile.EapType
            $This.ServerNames       = $xProfile.ServerNames
            $This.TrustedRootCA     = $xProfile.TrustedRootCA
            $This.Xml               = $xProfile.Xml
        }
        [Object] Profile()
        {
            $Object = @( )
            ForEach ($Item in "Password ConnectHiddenSSID EapType ServerNames TrustedRootCa Xml" -Split " ")
            {
                $Object += [WifiProfileExtensionProperty]::New($Object.Count,$Item,$This.$Item)
            }
            Return $Object
        }
        Full()
        {
            $This | Select-Object Index, Name, Guid, Description, IfIndex, Status, MacAddress,
            LinkSpeed, State, ProfileName, ConnectionMode, Authentication, Encryption, Password,
            ConnectHiddenSSID, EapType, ServerNames, TrustedRootCA, Xml
        }
    }
```

_____/ WifiProfileExtension

/_____ WifiProfileList /

This is specifically meant to organize the information for the 1) INTERFACE, 2) [WiFiProfile], and the 3) [WifiProfileExtension] information. It doesn't LOOK like it does a whole lot, but it's DOING something rather critical and important.

Effectively, this retains the information for the interface, and then as each profile is detected on that particular interface, the outside class can use a method to inject the output from wlanapi.dll... and then when that information is deposited into the list...? THEN, they can each be expanded into the full output.

```
    Class WifiProfileList
    {
```

```
        [Object] $Interface
        [Object] $Process
        [Object] $Output
        WifiProfileList([Object]$Interface)
        {
            $This.Interface = $Interface
            $This.Process   = @( )
            $This.Output    = @( )
        }
        Add([Object]$xProfile)
        {
            $This.Process  += [WifiProfile]::New($This.Process.Count,$xProfile)
        }
    }
```

```
                                                                                                                  _________________/
\_____________________________________________________________________________________________________________/ WifiProfileList
  WifiInterface /-------------------------------------------------------------------------------------------------------------------\
/---------------
```

I have a structural design implementation that I've been thinking about making to this particular class, which is the reason why I decided to write up this document. As I develop the graphical user interface, the things I wind up tweaking may actually reach back into ANY of those above classes, right...?

So, if I make a change to this particular class, then it MAY break something in the above classes if I do it incorrectly. And, that's bad. Because then, if people ARE using this tool in a production environment, or if it is used by penetration testers, or whatever...?

Then, it stands to reason that making structural design changes requires a fair amount of thought BEFORE haphazardly ripping apart something that already works. Because another solution would be to simply EXTEND the class with a separate class, somehow.

Weighing these options is akin to how a plumber may consider differing pipe materials on any particular job...

```
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
```

Plumber : Alright, so, I'm a plumber on a job... What am I gonna do...? Should I use PVC here...?
          It's upstate New York... pipes often FREEZE during the winter when they're not wrapped with heat tape.
          Doesn't matter what material it is, actually...
          But- PVC doesn't last nearly as long as copper. PVC is a lot less expensive than copper...
          Does the customer have the money for copper pipe...?
          What if there's something in the middle that'll work...? *shakes head* Hm...
          I'm a plumber, and sometimes I think of these very real situations from job to job...

```
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
```

```
    Class WifiInterface
    {
        [UInt32] $Index
        [String] $Name
        [Guid]   $Guid
        [String] $Description
        [UInt32] $IfIndex
        [String] $Status
        [String] $MacAddress
        [String] $LinkSpeed
        [String] $State
        [Object] $Profile
        [Object] $Connected
        WifiInterface([UInt32]$Index,[Object]$Interface)
        {
            $This.Index       = $Index
            $This.Name        = $Interface.Name
            $This.Guid        = $Interface.InterfaceGuid
            $This.Description = $Interface.InterfaceDescription
            $This.ifIndex     = $Interface.InterfaceIndex
            $This.Status      = $Interface.Status
            $This.MacAddress  = $Interface.MacAddress.Replace("-",":")
            $This.LinkSpeed   = $Interface.LinkSpeed
```

```
            $This.State        = Switch ($Interface.Status)
            {
                Up {"Connected"} Disconnected {"Disconnected"} Default {"Unknown"}
            }
            $This.Clear()
        }
        Add([Object]$xProfile)
        {
            $This.Profile.Add($xProfile)
        }
        Clear()
        {
            $This.Profile      = [WifiProfileList]::New($This)
        }
    }
```

```
                                                                                                        ________________/
\______________________________________________________________________________________________________/ WifiInterface
  WifiInterfaceNetsh /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

    This class DOES actually utilize netsh, and converts the information into an object that is strictly meant for
    determining whether the adapter is connected to a network, AND, the information that is returned, if it is.

    So, it'll return the (BSSID/MacAddress) of the radio that it is currently connected to.
    SOMETIMES, SSID's will be <HIDDEN>... then what...?
    How does one go about connecting to a hidden network...?

    Well, I can tell ya, NOBODY is allowed to summon a genie from a magic lamp, and say:

    /¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\

    Somebody : Alright, magic genie...
               I want my device to connect to THAT hidden network...
    Genie    : *scoffs* Buddy, I can't help ya with that.
    Somebody : ...are you serious...?
    Genie    : Yeh, dude.
               Gotta use a program to do THAT...
               Gotta use the IEEE standards, and some CLI program like netsh or whatever, to do THAT...
               *shakes head* Unbelievable.
               Guy summons me to have his device connect to a particular hidden network or whatever...?
               *eyebrows up* Wow...
               What's this world coming to...?

    \__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/

    Look, is that REALISTC...? Nah.
    But, what the genie says in the skit, about using a program to do that...?
    That part is rather realistic, after all.
```

```
    Class WifiInterfaceNetsh
    {
        [String] $Name
        [String] $Description
        [Guid]   $Guid
        [String] $MacAddress
        [String] $InterfaceType
        [String] $State
        [String] $Ssid
        [String] $Bssid
        [String] $NetworkType
        [String] $RadioType
        [String] $Authentication
        [String] $Cipher
        [String] $Connection
        [String] $Band
        [UInt32] $Channel
        [Float]  $Receive
        [Float]  $Transmit
        [String] $Signal
```

```
        [String] $Profile
        WifiInterfaceNetsh([String[]]$In)
        {
            $This.Name                  = $This.Tx($In,"Name")
            $This.Description           = $This.Tx($In,"Description")
            $This.GUID                  = $This.Tx($In,"GUID")
            $This.MacAddress            = $This.Tx($In,"Physical address")
            $This.InterfaceType         = $This.Tx($In,"Interface type")
            $This.State                 = $This.Tx($In,"State")
            $This.Ssid                  = $This.Tx($In,"SSID")
            $This.Bssid                 = $This.Tx($In,"BSSID") | % ToUpper
            $This.NetworkType           = $This.Tx($In,"Network type")
            $This.RadioType             = $This.Tx($In,"Radio type")
            $This.Authentication        = $This.Tx($In,"Authentication")
            $This.Cipher                = $This.Tx($In,"Cipher")
            $This.Connection            = $This.Tx($In,"Connection mode")
            $This.Band                  = $This.Tx($In,"Band")
            $This.Channel               = $This.Tx($In,"Channel")
            $This.Receive               = $This.Tx($In,"Receive rate \(Mbps\)")
            $This.Transmit              = $This.Tx($In,"Transmit rate \(Mbps\)")
            $This.Signal                = $This.Tx($In,"Signal")
            $This.Profile               = $This.Tx($In,"Profile")
        }
        [String] Tx([Object]$In,[String]$String)
        {
            Return $In | ? { $_ -match "(^\s+$String\s+\:)" } | % Substring 29
        }
    }
```

```
                                                                                                                    ____________________/
\___________________________________________________________________________________________________________________/ WifiInterfaceNetsh
  WifiProfileSubcontroller /-------------------------------------------------------------------------------------------------------------\
/--------------------------
```

This class effectively controls all of the enumeration (types/lists), and it is the other half of the [WifiProfileList] class up above. When this class is instantiated, it prepares the enumeration types.

Then, when each adapter is polled for it's profiles and whatnot...?
The [WifiProfileList] class will collect the information from the “wlanapi.dll” file, and then the method [Object] Load([UInt32]$Index,[Object]$Interface,[Object]$xProfile) will combine all of the information by creating a template that uses the [WifiProfileExtension] class.

This whole process is actually incredibly complicated to conceptualize without some sort of example.
The objective is to only pull the information it needs ONCE, and then use that information as a reference.
That's not a whole lot different from how Assembly code works.

In Assembly, when programming code is compiled, it uses [IntPtr] (objects/references) to store things in address spaces and such. I won't talk about Assembly, because it is actually rather complex... but, how it relates to [PowerShell], is that to improve the efficiency of the program, don't replicate the same information thousands or millions of times. That's a pretty great way to increase the efficiency of a program.

But, these pointers all refer to a place that is in memory.
Rather than to have a class that has ALL of that information in each object that gets instantiated...?
Replicate certain properties and use those properties and methods as a reference, so that the entire TREE isn't replicated, but rather, the desired value from the enumeration tree.

That is exactly what this approach is doing, while also allowing additional (throttling/control) points.
So, below in this class there are a couple of switches that are looking for specific information to come back from the profile objects.

If the [ProfileXml] object says “Open” for its' authentication method...? That means “Open80211”.
If it says “WPA2PSK”, that means “WpaPsk”.
Some of these terms are things that the IEEE looks for ahead of time when they develop standards and stuff.
However, many of the terms eventually overlap with one another.

Also, if the [ProfileXml] object says “AES” for its' encryption method...? That means “Ccmp”.
Typically, that is exactly what that means, and it is used quite extensively.

```
    Class WifiProfileSubcontroller
    {
```

```
        [Object] $Connection
        [Object] $Authentication
        [Object] $Encryption
        WifiProfileSubcontroller()
        {
            $This.Connection         = [ConnectionModeList]::New()
            $This.Authentication     = [AuthenticationList]::New()
            $This.Encryption         = [EncryptionList]::New()
        }
        [Object] Load([UInt32]$Index,[Object]$Interface,[Object]$xProfile)
        {
            $Template                = [WifiProfileExtension]::New($Interface)
            $Template.Index          = $Index
            $Template.ConnectionMode = $This.Connection.Get($xProfile.Detail.ConnectionMode)

            Switch -Regex ($xProfile.Detail.Authentication)
            {
                ^Open$   { $xProfile.Detail.Authentication = "Open80211" }
                ^WPA2PSK { $xProfile.Detail.Authentication = "WpaPsk"    }
                Default  { }
            }

            $Template.Authentication = $This.Authentication.Get($xProfile.Detail.Authentication)

            Switch -Regex ($xProfile.Detail.Encryption)
            {
                ^AES$    { $xProfile.Detail.Encryption    = "Ccmp"       }
                Default  { }
            }

            $Template.Encryption     = $This.Encryption.Get($xProfile.Detail.Encryption)

            $Template.Load($xProfile.Detail)
            Return $Template
        }
    }
```

_____/
_____/ WifiProfileSubcontroller
RtMethod /_____\
/----------

This class is specifically meant for obtaining the [System.Threading.Tasks.Task`1] object back from the running runtime type, [System.WindowsRuntimeSystemExtensions]. Here's the thing.

In [C#], it is actually really easy to access that method by using Task[<object>].
In [PowerShell], uh- it takes a little bit of finagling, to get the correct method.

This particular class isn't really doing a whole lot...
...other than being a part of a method in the [WirelessController] class further below.

```
    Class RtMethod
    {
        [String] $Name
        [Object] $Params
        [Object] $Count
        [Object] $Object
        RtMethod([Object]$Object)
        {
            $This.Object = $Object
            $This.Params = $Object.GetParameters()
            $This.Count  = $This.Params.Count
            $This.Name   = $This.Params[0].ParameterType.Name
        }
    }
```

_____/
_____/ RtMethod
WirelessSubcontroller /_____\

/-----------------------

    This particular class is strictly meant for organizing information related to either (adapter(s)/interface(s)),
    OR, (network(s)/ssid(s)).

    The reason it exists, is mainly because it provides a way to control these objects by having specific names and
    properties left in like, their own (domain/namespace) so to speak.

    Think of it like this.
    If someone puts all of their documents, pictures, videos, downloads, desktop files into the SAME FOLDER...?
    Then, that means you'll have to like, search through ALL of those files each time you want to access any
    specific file. That makes the process more tedious.

    Whereas, separating each of those file types allows a person to double-click the pictures folder, to look for
    a certain picture file. Or, et cetera.

    That's the idea here.

```
Class WirelessSubcontroller
{
    [String]     $Type
    [Object]     $List
    [UInt32]    $Total
    [Int32]     $Index
    WirelessSubcontroller([String]$Type)
    {
        $This.Type     = $Type
        $This.Clear()
    }
    Clear()
    {
        $This.List     = @( )
        $This.Total    = 0
        $This.Index    = -1
    }
    Add([Object]$Item)
    {
        $This.List    += $Item
        $This.Total   ++
    }
    Select([UInt32]$Index)
    {
        If ($Index -gt $This.Total)
        {
            Throw "Invalid index"
        }

        $This.Index    = $Index
    }
    Unselect()
    {
        $This.Index    = -1
    }
    [Object] Selected()
    {
        If ($This.Index -eq -1)
        {
            Throw "No adapter selected"
        }

        Return $This.List[$This.Index]
    }
}
```

                                                                                                                  _____________________/
____/ WirelessSubcontroller
  WirelessController /---\
/--------------------

    Ok, so this is where all of the above information is going to be controlled.

Since the last time I made the video featured in the beginning of this document in the script portion,
I've added some functionality that allows for this class to be controlled from the CLI as well as the GUI.

The reason being, suppose I really DO wanna do this stuff from the command line, could anybody do that...?
If they can, does that mean somebody's gonna type ALL of that stuff jcwalker hammered out, on his Github...?
I mean, suppose that someone was patient enough to do that, can somebody do all of that from like, DOS...?
Imagine if you could install DOS, and then access a wireless access point from the DOS command prompt...

```
Dos:\> Connect-To-Wireless-Access-Point
Dos:\> ERROR: I DON'T KNOW WHAT THAT EVEN MEANS, BRO... BYE.
```

Nah. Pretty sure nobody's gonna do that. That's why programs exist.
Netsh is probably the closest thing that anybody's gonna get to doing just that, but even still...
Netsh isn't going to return the extensive amount of information that THIS utility, in conjunction with jcwalker's stuff, will return from wlanapi.dll.

But then ALSO making it easy to control from a graphical user interface...?
There's a lot of complexity to this utility, and making it hasn't been all that easy.

There are plenty of programs out there that do what this program does...
But, most of those programs out there in the wild were made by TEAMS of people.

```
Class WirelessController
{
    Hidden [UInt32]     $Mode
    Hidden [Object]    $Module
    Hidden [String]   $OEMLogo
    Hidden [Object]     $Ssid
    Hidden [Object]   $Profile
    Hidden [Object]     $Xaml
    [Object]          $Adapter
    Hidden [Object]   $Request
    Hidden [Object]    $Radios
    Hidden [Object]     $List
    [Object]          $Network
    [Object]        $Connected
    WirelessController([UInt32]$Mode)
    {
        $This.Mode     = $Mode

        # // ____________________________
        # // | Load the module location |
        # // ----------------------------

        $This.Module   = Get-FEModule -Mode 1
        If (!$This.Module)
        {
            Throw "Must install [FightingEntropy($([Char]960))]"
        }

        $This.OEMLogo  = $This.Module._Graphic("OEMLogo.bmp").Fullname

        # // ________________________________________
        # // | Load the wireless profile type/object |
        # // ----------------------------------------

        Add-Type -Path $This.Module._Control("Wifi.cs").Fullname -ErrorAction Continue

        # // ______________________________________
        # // | Load the Ssid+Profile subcontrollers |
        # // --------------------------------------

        $This.Ssid      = $This.GetSsidSubController()
        $This.Profile   = $This.GetProfileSubcontroller()

        # // ___________________________________________
        # // | Load the adapter/+network subcontrollers |
        # // -------------------------------------------

        $This.Adapter   = $This.GetAdapterSubcontroller()
```

```
            $This.Network   = $This.GetNetworkSubcontroller()

            # // ______________________________________
            # // | Prime the radio task(s) init state |
            # // --------------------------------------

            $This.Request   = @( )
            $This.Radios    = @( )
            $This.List      = @( )

            # // ____________________________________________________________
            # // | Load all available wireless adapters into the adapter list |
            # // ------------------------------------------------------------

            $This.RefreshWirelessAdapterList()

            # // ___________________________________
            # // | Throw if no existing wireless adapters |
            # // -----------------------------------

            If ($This.Adapter.Count -eq 0)
            {
                Throw "No existing wireless adapters on this system"
            }

            If ($This.Mode -eq 1)
            {
                $This.Xaml = $This.GetWirelessNetworkXaml()

                $This.Xaml.IO.Adapter.Items.Clear()
                $This.Xaml.IO.Profile.Items.Clear()
                $This.Xaml.IO.ProfileExtension.Items.Clear()
                $This.Xaml.IO.Network.Items.Clear()

                # // _________________________
                # // | Populate adapter box |
                # // -------------------------

                ForEach ($Adapter in $This.Adapter.List)
                {
                    $This.Xaml.IO.Adapter.Items.Add($Adapter)
                }

                # // ________________________________________
                # // | Set other various starting conditions |
                # // ----------------------------------------

                If ($This.Xaml.IO.Adapter.Count -gt 0)
                {
                    $This.Xaml.IO.Refresh.IsEnabled = 1
                }

                $This.StageXamlEvent()
            }
        }
        [Object] GetSsidSubcontroller()
        {
            Return [SsidSubcontroller]::New()
        }
        [Object] GetProfileSubcontroller()
        {
            Return [WifiProfileSubcontroller]::New()
        }
        [Object] GetAdapterSubcontroller()
        {
            Return [WirelessSubcontroller]::New("Adapter")
        }
        [Object] GetNetworkSubcontroller()
        {
            Return [WirelessSubcontroller]::New("Network")
        }
        [Object] GetWirelessNetworkXaml()
```

```
        {
            Return [XamlWindow][WirelessNetworkXaml]::Content
        }
        [Object] GetPassphraseXaml()
        {
            Return [XamlWindow][PassphraseXaml]::Content
        }
        [Object] Task()
        {
            Return [System.WindowsRuntimeSystemExtensions].GetMethods() | ? Name -eq AsTask | % {

                [RtMethod]$_

            } | ? Count -eq 1 | ? Name -eq IAsyncOperation``1 | % Object
        }
        [Object] RxStatus()
        {
            Return [Windows.Devices.Radios.RadioAccessStatus]
        }
        [Object[]] RxAsync()
        {
            Return [Windows.Devices.Radios.Radio]::RequestAccessAsync()
        }
        [Object] RsList()
        {
            Return [System.Collections.Generic.IReadOnlyList[Windows.Devices.Radios.Radio]]
        }
        [Object[]] RsAsync()
        {
            Return [Windows.Devices.Radios.Radio]::GetRadiosAsync()
        }
        [Object] RaList()
        {
            Return [System.Collections.Generic.IReadOnlyList[Windows.Devices.WiFi.WiFiAdapter]]
        }
        [Object[]] RaAsync()
        {
            Return [Windows.Devices.WiFi.WiFiAdapter]::FindAllAdaptersAsync()
        }
        [Object] RadioRequestAccess()
        {
            Return $This.Task().MakeGenericMethod($This.RxStatus()).Invoke($Null,$This.RxAsync())
        }
        [Object] RadioSynchronization()
        {
            Return $This.Task().MakeGenericMethod($This.RsList()).Invoke($Null, $This.RsAsync())
        }
        [Object] RadioFindAllAdaptersAsync()
        {
            Return $This.Task().MakeGenericMethod($This.RaList()).Invoke($Null, $This.RaAsync())
        }
        [Object] NetshShowInterface([String]$Name)
        {
            Return [WifiInterfaceNetsh]::New((netsh wlan show interface $Name))
        }
        RefreshWirelessAdapterList()
        {
            $This.Adapter.Clear()
            ForEach ($Adapter in Get-NetAdapter | ? PhysicalMediaType -match "(Native 802.11|Wireless (W|L)AN)")
            {
                $Item          = [WifiInterface]::New($This.Adapter.Total,$Adapter)
                $This.GetWiFiProfileList($Item)
                $This.Adapter.Add($Item)
            }
        }
        [Object] Win32Exception([UInt32]$ReasonCode)
        {
            Return [System.ComponentModel.Win32Exception]::New($ReasonCode)
        }
        [Object] WlanProfileInfoObject()
        {
            Return New-Object WiFi.ProfileManagement+ProfileInfo
```

```
        }
        [Object] WlanConnectionParams()
        {
            Return New-Object WiFi.ProfileManagement+WLAN_CONNECTION_PARAMETERS
        }
        [Object] WlanConnectionMode([String]$ConnectionMode)
        {
            Return (New-Object WiFi.ProfileManagement+WLAN_CONNECTION_MODE)::$ConnectionMode
        }
        [Object] WlanDot11BssType([String]$Dot11BssType)
        {
            Return (New-Object WiFi.ProfileManagement+DOT11_BSS_TYPE)::$Dot11BssType
        }
        [Object] WlanConnectionFlag([String]$Flag)
        {
            Return (New-Object WiFi.ProfileManagement+WlanConnectionFlag)::$Flag
        }
        [Object] WlanSetProfile([UInt32]$Handle,[Guid]$IFGuid,[UInt32]$Flags,[IntPtr]$ProfileXml,
        [IntPtr] $ProfileSecurity,[Bool]$Overwrite,[IntPtr]$pReserved,[IntPtr]$pdwReasonCode)
        {
            Return (New-Object WiFi.ProfileManagement)::WlanSetProfile($Handle,$IFGuid,$Flags,
            $ProfileXml,$ProfileSecurity,$Overwrite,$pReserved,$pdwReasonCode)
        }
        [Void] WlanDeleteProfile([IntPtr]$Handle,[Guid]$IFGuid,[String]$ProfileID,[IntPtr]$pReserved)
        {
            (New-Object WiFi.ProfileManagement)::WlanDeleteProfile($Handle,$IFGuid,$ProfileID,$pReserved)
        }
        [Void] WlanDisconnect([IntPtr]$Handle,[Guid]$IFGuid,[IntPtr]$pReserved)
        {
            (New-Object WiFi.ProfileManagement)::WlanDisconnect($Handle,$IFGuid,$pReserved)
        }
        [Void] WlanConnect([IntPtr]$Handle,[Guid]$IFGuid,[Object]$Params,[IntPtr]$pReserved)
        {
            (New-Object WiFi.ProfileManagement)::WlanConnect($Handle,$IFGuid,$Params,$pReserved)
        }
        [String] WiFiReasonCode([IntPtr]$Reason)
        {
            $String = [Text.StringBuilder]::New(1024)
            $Result = (New-Object WiFi.ProfileManagement)::WlanReasonCodeToString(
                        $Reason.ToInt32(),
                        $String.Capacity,
                        $String,
                        [IntPtr]::Zero)

            If ($Result -ne 0)
            {
                Return $This.Win32Exception($Result)
            }

            Return $String.ToString()
        }
        [IntPtr] NewWifiHandle()
        {
            $Max       = 2
            [Ref] $Neg = 0
            $Handle    = [IntPtr]::Zero
            $Result    = (New-Object WiFi.ProfileManagement)::WlanOpenHandle(
                            $Max,
                            [IntPtr]::Zero,
                            $Neg,
                            [Ref]$Handle)

            If ($Result -eq 0)
            {
                Return $Handle
            }
            Else
            {
                Throw $This.Win32Exception($Result)
            }
        }
        [Void] RemoveWifiHandle([IntPtr]$Handle)
```

```
        {
            $Result = (New-Object WiFi.ProfileManagement)::WlanCloseHandle($Handle,[IntPtr]::Zero)
            If ($Result -ne 0)
            {
                $Message = $This.Win32Exception($Result)
                Throw "$Message / $Result"
            }
        }
        GetWiFiProfileList([Object]$Adapter)
        {
            $Ptr       = 0
            $Handle    = $This.NewWifiHandle()

            # // ------------------------------------------
            # // | Get the profile list, save to pointer |
            # // ------------------------------------------

            [Void](New-Object WiFi.ProfileManagement)::WlanGetProfileList(
                      $Handle,
                      $Adapter.Guid,
                      [IntPtr]::Zero,
                      [Ref]$Ptr)

            # // ------------------------------------------
            # // | Process all profiles for this adapter |
            # // ------------------------------------------

            (New-Object WiFi.ProfileManagement+WLAN_PROFILE_INFO_LIST $Ptr).ProfileInfo | % { $Adapter.Add($_) }

            $This.RemoveWiFiHandle($Handle)

            # // ----------------------------------
            # // | Obtain details for each profile |
            # // ----------------------------------

            ForEach ($X in 0..($Adapter.Profile.Process.Count-1))
            {
                $xProfile               = $Adapter.Profile.Process[$X]
                [IntPtr]$Handle         = $This.NewWifiHandle()
                $Flags                  = 0
                $xProfile.Detail        = $This.WiFiProfileInfo($xProfile.Name,$Adapter.Guid,$Handle,$Flags)
                $This.RemoveWiFiHandle($Handle)

                $Adapter.Profile.Output += $This.Profile.Load($X,$Adapter,$xProfile)
            }
        }
        [Object] WiFiProfileInfo([String]$Tag,[Guid]$Guid,[IntPtr]$Handle,[Int16]$Flags)
        {
            [String] $pstrXml  = $Null
            $WlanAccess        = 0
            $WlanPF            = $Flags
            $Result            = (New-Object WiFi.ProfileManagement)::WlanGetProfile(
                                  $Handle,$Guid,$Tag,[IntPtr]::Zero,
                                  [Ref]$pstrXml,[Ref]$WlanPF,[Ref]$WlanAccess)
            $Password          = $Null
            $ConnectHiddenSSID = $Null
            $EapType           = $Null
            $XmlPtr            = $Null
            $ServerNames       = $Null
            $RootCA            = $Null
            $Return            = $Null

            If ($Result -ne 0)
            {
                Return $This.Win32Exception($Result)
            }

            $WlanProfile       = [Xml]$pstrXml

            # // -------------------
            # // | Parse password |
            # // -------------------
```

```powershell
        If ($Flags -eq 13)
        {
            $Password      = $WlanProfile.WlanProfile.Msm.Security.SharedKey.KeyMaterial
        }
        If ($Flags -ne 13)
        {
            $Password      = $Null
        }

        # // ____________________________
        # // | Parse nonBroadcast flag |
        # // ----------------------------

        If ([Bool]::TryParse($WlanProfile.WlanProfile.SsidConfig.NonBroadcast,[Ref]$Null))
        {
            $ConnectHiddenSSID = [Bool]::Parse($WlanProfile.WlanProfile.SsidConfig.NonBroadcast)
        }
        Else
        {
            $ConnectHiddenSSID = $false
        }

        # // __________________
        # // | Parse EAP type |
        # // ------------------

        If ($WlanProfile.WlanProfile.Msm.Security.AuthEncryption.UseOneX -eq $true)
        {
            $WlanProfile.WlanProfile.Msm.Security.OneX.EapConfig.EapHostConfig.EapMethod.Type.InnerText | % {

                $EAPType = Switch ($_)
                {
                    13      {    'TLS'  } # EAP-TLS
                    25      {    'PEAP' } # EAP-PEAP (MSCHAPv2)
                    Default { 'Unknown' }
                }
            }
        }
        Else
        {
            $EAPType = $Null
        }

        # // _______________________________
        # // | Parse Validation Server Name |
        # // -------------------------------

        If (!!$EapType)
        {
            $Config = $WlanProfile.WlanProfile.Msm.Security.OneX.EapConfig.EapHostConfig.Config
            Switch ($EapType)
            {
                PEAP
                {

                    $ServerNames = $Config.Eap.EapType.ServerValidation.ServerNames
                }

                TLS
                {
                    $Node        = $Config.SelectNodes("//*[local-name()='ServerNames']")
                    $ServerNames = $Node[0].InnerText
                }
            }
        }

        # // _________________________________
        # // | Parse Validation TrustedRootCA |
        # // ---------------------------------

        If (!!$EAPType)
```

```
        {
            $Config = $WlanProfile.WlanProfile.Msm.Security.OneX.EapConfig.EapHostConfig.Config
            Switch ($EAPType)
            {
                PEAP
                {
                    $RootCA = $Config.Eap.EapType.ServerValidation.TrustedRootCA.Replace(' ','') | % ToLower
                }
                TLS
                {
                    $Node   = $Config.SelectNodes("//*[local-name()='TrustedRootCA']")
                    $RootCA = $Node[0].InnerText.Replace(' ','') | % ToLower
                }
            }
        }

        $Return                    = $This.WlanProfileInfoObject()
        $Return.ProfileName        = $WlanProfile.WlanProfile.SsidConfig.Ssid.name
        $Return.ConnectionMode     = $WlanProfile.WlanProfile.ConnectionMode
        $Return.Authentication     = $WlanProfile.WlanProfile.Msm.Security.AuthEncryption.Authentication
        $Return.Encryption         = $WlanProfile.WlanProfile.Msm.Security.AuthEncryption.Encryption
        $Return.Password           = $Password
        $Return.ConnectHiddenSSID  = $ConnectHiddenSSID
        $Return.EAPType            = $EAPType
        $Return.ServerNames        = $ServerNames
        $Return.TrustedRootCA      = $RootCA
        $Return.Xml                = $pstrXml

        $XmlPtr                    = [System.Runtime.InteropServices.Marshal]::StringToHGlobalAuto($pstrXml)
        (New-Object WiFi.ProfileManagement)::WlanFreeMemory($XmlPtr)

        Return $Return
    }
    [Object] GetWiFiProfileInfo([String]$Tag,[Guid]$Guid,[Int16]$Flags)
    {
        [IntPtr]$Handle = $This.NewWifiHandle()
        $WlanFlags      = $Flags
        $Return         = $This.WiFiProfileInfo($Tag,$Guid,$Handle,$WlanFlags)
        $This.RemoveWiFiHandle($Handle)
        Return $Return
    }
    [Object] GetWifiProfileInfo([String]$Tag,[Guid]$Guid)
    {
        [IntPtr]$Handle = $This.NewWifiHandle()
        $WlanFlags      = 0
        $Return         = $This.WiFiProfileInfo($Tag,$Guid,$Handle,$WlanFlags)
        $This.RemoveWiFiHandle($Handle)
        Return $Return
    }
    [Object] GetWiFiConnectionParameter([String]$Tag,[String]$Mode,[String]$Type,[String]$Flag)
    {
        Return $This.WifiConnectionParameter($Tag,$Mode,$Type,$Flag)
    }
    [Object] GetWiFiConnectionParameter([String]$Tag,[String]$Mode,[String]$Type)
    {
        Return $This.WifiConnectionParameter($Tag,$Mode,$Type,"Default")
    }
    [Object] GetWiFiConnectionParameter([String]$Tag,[String]$Mode)
    {
        Return $This.WifiConnectionParameter($Tag,$Mode,"Any","Default")
    }
    [Object] GetWiFiConnectionParameter([String]$Tag)
    {
        Return $This.WifiConnectionParameter($Tag,"Profile","Any","Default")
    }
    [Object] WifiConnectionParameter([String]$Tag,[String]$Mode,[String]$Type,[String]$Flag)
    {
        Try
        {
            $Resolver                  = [ConnectionModeResolver]::New()
            $Connect                   = $This.WlanConnectionParams()
            $Connect.StrProfile        = $Tag
```

```
            $Connect.WlanConnectionMode = $This.WlanConnectionMode($Resolver.$Mode)
            $Connect.Dot11BssType       = $This.WlanDot11BssType($Type)
            $Connect.dwFlags            = $This.WlanConnectionFlag($Flag)
        }
        Catch
        {
            Throw "An error occurred while setting connection parameters"
        }

        Return $Connect
    }
    [Object] FormatXml([Object]$Content)
    {
        $Str             = [System.IO.StringWriter]::New()
        $Xml             = [System.Xml.XmlTextWriter]::New($Str)
        $Xml.Formatting  = "Indented"
        $Xml.Indentation = 4
        ([Xml]$Content).WriteContentTo($Xml)
        $Xml.Flush()
        $Str.Flush()
        Return $Str.ToString()
    }
    [Object] XmlTemplate([UInt32]$Type)
    {
        $xList = (0,"Personal"),(1,"EapPeap"),(2,"EapTls") | % { "($($_[0]): $($_[1]))" }

        If ($Type -notin 0..2)
        {
            Throw "Select a valid type: [$($xList -join ", ")]"
        }

        $P = "http://www.microsoft.com/provisioning"

        $xProfile = Switch ($Type)
        {
            0 # WiFiProfileXmlPersonal
            {
                '<?xml version="1.0"?>',('<WLANProfile xmlns="http://www.microsoft.com/networking/WLAN/pr'+
                'ofile/v1">'),'<name>{0}</name>','<SSIDConfig>','<SSID>','<hex>{1}</hex>',('<name>{0}</na'+
                'me>'),'</SSID>','</SSIDConfig>','<connectionType>ESS</connectionType>',('<connectionMode'+
                '>{2}</connectionMode>'),'<MSM>','<security>','<authEncryption>',('<authentication>{3}</a'+
                'uthentication>'),'<encryption>{4}</encryption>','<useOneX>false</useOneX>',('</authEncry'+
                'ption>'),'<sharedKey>','<keyType>passPhrase</keyType>','<protected>false</protected>',
                '<keyMaterial>{5}</keyMaterial>','</sharedKey>','</security>','</MSM>',('<MacRandomizatio'+
                'n xmlns="http://www.microsoft.com/networking/WLAN/profile/v3">'),('<enableRandomization>'+
                'false</enableRandomization>'),"</MacRandomization>",'</WLANProfile>'
            }
            1 # WiFiProfileXmlEapPeap
            {
                '<?xml version="1.0"?>',('<WLANProfile xmlns="http://www.microsoft.com/networking/WLAN/pr'+
                'ofile/v1">'),'<name>{0}</name>','<SSIDConfig>','<SSID>','<hex>{1}</hex>',('<name>{0}</na'+
                'me>'),'</SSID>',('</SSIDConfig>'),'<connectionType>ESS</connectionType>',('<connectionMo'+
                'de>{2}</connectionMode>'),'<MSM>','<security>','<authEncryption>',('<authentication>{3}<'+
                '/authentication>'),'<encryption>{4}</encryption>','<useOneX>true</useOneX>',('</authEncr'+
                'yption>'),'<PMKCacheMode>enabled</PMKCacheMode>','<PMKCacheTTL>720</PMKCacheTTL>',('<PMK'+
                'CacheSize>128</PMKCacheSize>'),'<preAuthMode>disabled</preAuthMode>',('<OneX xmlns="http'+
                '://www.microsoft.com/networking/OneX/v1">'),'<authMode>machineOrUser</authMode>',('<EAPC'+
                'onfig>'),"<EapHostConfig xmlns='$P/EapHostConfig'>",'<EapMethod>',("<Type xmlns='$P/EapH"+
                "ostConfig'>25</Type>"),"<VendorId xmlns='$P/EapCommon'>0</VendorId>",("<VendorType xmlns"+
                "='$P/EapCommon'>0</VendorType>"),"<AuthorId xmlns='$P/EapCommon'>0</AuthorId>",('</EapMe'+
                'thod>'),"<Config xmlns='$P/EapHostConfig'>",("<Eap xmlns='$P/BaseEapConnectionProperties"+
                "V1'>"),'<Type>25</Type>',"<EapType xmlns='$P/MsPeapConnectionPropertiesV1'>",('<ServerVa'+
                'lidation>'),('<DisableUserPromptForServerValidation>false</DisableUserPromptForServerVal'+
                'idation>'),'<ServerNames></ServerNames>','<TrustedRootCA></TrustedRootCA>',('</ServerVal'+
                'idation>'),'<FastReconnect>true</FastReconnect>',('<InnerEapOptional>false</InnerEapOpti'+
                'onal>'),"<Eap xmlns='$P/BaseEapConnectionPropertiesV1'>",'<Type>26</Type>',("<EapType xm"+
                "lns='$P/MsChapV2ConnectionPropertiesV1'>"),('<UseWinLogonCredentials>false</UseWinLogonC'+
                'redentials>'),'</EapType>','</Eap>',('<EnableQuarantineChecks>false</EnableQuarantineChe'+
                'cks>'),'<RequireCryptoBinding>false</RequireCryptoBinding>','<PeapExtensions>',("<Perfor"+
                "mServerValidation xmlns='$P/MsPeapConnectionPropertiesV2'>true</PerformServerValidation>"+
                ""),"<AcceptServerName xmlns='$P/MsPeapConnectionPropertiesV2'>true</AcceptServerName>",
                "<PeapExtensionsV2 xmlns='$P/MsPeapConnectionPropertiesV2'>",("<AllowPromptingWhenServerC"+
```

```
                "ANotFound xmlns='$P/MsPeapConnectionPropertiesV3'>true</AllowPromptingWhenServerCANotFou"+
                "nd>"),'</PeapExtensionsV2>','</PeapExtensions>','</EapType>','</Eap>','</Config>',('</Ea'+
                'pHostConfig>'),'</EAPConfig>','</OneX>','</security>','</MSM>',('<MacRandomization xmlns'+
                '="http://www.microsoft.com/networking/WLAN/profile/v3">'),("<enableRandomization>false</"+
                "enableRandomization>"),"</MacRandomization>",'</WLANProfile>'
            }
            2 # WiFiProfileXmlEapTls
            {
                '<?xml version="1.0"?>',('<WLANProfile xmlns="http://www.microsoft.com/networking/WLAN/pr'+
                'ofile/v1">'),'<name>{0}</name>','<SSIDConfig>','<SSID>','<hex>{1}</hex>',('<name>{0}</na'+
                'me>'),'</SSID>','</SSIDConfig>','<connectionType>ESS</connectionType>',('<connectionMode'+
                '>{2}</connectionMode>'),'<MSM>','<security>','<authEncryption>',('<authentication>{3}</a'+
                'uthentication>'),'<encryption>{4}</encryption>','<useOneX>true</useOneX>',('</authEncryp'+
                'tion>'),'<PMKCacheMode>enabled</PMKCacheMode>','<PMKCacheTTL>720</PMKCacheTTL>',('<PMKCa'+
                'cheSize>128</PMKCacheSize>'),'<preAuthMode>disabled</preAuthMode>',('<OneX xmlns="http:/'+
                '/www.microsoft.com/networking/OneX/v1">'),'<authMode>machineOrUser</authMode>',('<EAPCon'+
                'fig>'),"<EapHostConfig xmlns='$P/EapHostConfig'>",'<EapMethod>',("<Type xmlns='$P/EapHos"+
                "tConfig'>13</Type>"),"<VendorId xmlns='$P/EapCommon'>0</VendorId>",("<VendorType xmlns='"+
                "$P/EapCommon'>0</VendorType>"),"<AuthorId xmlns='$P/EapCommon'>0</AuthorId>",('</EapMeth'+
                'od>'),("<Config xmlns:baseEap='$P/BaseEapConnectionPropertiesV1' xmlns:eapTls='$P/EapTls"+
                "ConnectionPropertiesV1'>"),'<baseEap:Eap>','<baseEap:Type>13</baseEap:Type>',('<eapTls:E'+
                'apType>'),'<eapTls:CredentialsSource>','<eapTls:CertificateStore />',('</eapTls:Credenti'+
                'alsSource>'),'<eapTls:ServerValidation>',('<eapTls:DisableUserPromptForServerValidation>'+
                'false</eapTls:DisableUserPromptForServerValidation>'),('<eapTls:ServerNames></eapTls:Ser'+
                'verNames>'),'<eapTls:TrustedRootCA></eapTls:TrustedRootCA>','</eapTls:ServerValidation>',
                '<eapTls:DifferentUsername>false</eapTls:DifferentUsername>','</eapTls:EapType>',('</base'+
                'Eap:Eap>'),'</Config>','</EapHostConfig>','</EAPConfig>','</OneX>','</security>','</MSM>',
                '<MacRandomization xmlns="http://www.microsoft.com/networking/WLAN/profile/v3">',("<enabl"+
                "eRandomization>false</enableRandomization>"),"</MacRandomization>",'</WLANProfile>'
            }
        }

        Return $This.FormatXml($xProfile)
    }
    [String] Hex([String]$Tag)
    {
        Return ([Char[]]$Tag | % { '{0:X}' -f [Int]$_ }) -join ''
    }
    [String] NewWiFiProfileXmlPsk([String]$Tag,[String]$Mode='Auto',[String]$Auth='WPA2PSK',
    [String]$Enc='AES',[SecureString]$Pass)
    {
        $Plain          = $Null
        $ProfileXml     = $Null
        $Hex            = $This.Hex($Tag)
        Try
        {
            If ($Pass)
            {
                $Secure = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($Pass)
                $Pass   = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($Secure)
            }

            $ProfileXml = [XML]($This.XmlTemplate(0) -f $Tag, $Hex, $Mode, $Auth, $Enc, $Plain)
            If (!$Plain)
            {
                $ProfileXml.WlanProfile.Msm.Security | % { $Null = $_.RemoveChild($_.SharedKey) }
            }

            If ($Auth -eq 'WPA3SAE')
            {
                # // ------------------------------------------
                # // | Set transition mode as true for WPA3-SAE |
                # // ------------------------------------------

                $Names   = [System.Xml.XmlNamespaceManager]::new($ProfileXml.NameTable)
                $Names.AddNamespace('WLANProfile',$ProfileXml.DocumentElement.GetAttribute('xmlns'))
                $RefNode = $ProfileXml.SelectSingleNode('//WLANProfile:authEncryption', $Names)
                $XmlNode = $ProfileXml.CreateElement(
                           'transitionMode',
                           'http://www.microsoft.com/networking/WLAN/profile/v4')
                $XmlNode.InnerText = 'True'
                $Null    = $RefNode.AppendChild($XmlNode)
```

```
            }

            Return $This.FormatXml($ProfileXml.OuterXml)
        }
        Catch
        {
            Throw "Failed to create a new profile"
        }
    }
    [String] NewWifiProfileXmlEap([String]$Tag,[String]$Mode='Auto',[String]$Auth='WPA2PSK',
    [String]$Enc='AES',[String]$Eap,[String[]]$ServerNames,[String]$RootCA)
    {
        $ProfileXml = $Null
        $Hex        = $This.Hex($Tag)
        Try
        {
            If ($Eap -eq 'PEAP')
            {
                $ProfileXml = [Xml]($This.XmlTemplate(1) -f $Tag, $Hex, $Mode, $Auth, $Enc)
                $Config = $ProfileXml.WlanProfile.Msm.Security.OneX.EapConfig.EapHostConfig.Config

                If ($ServerNames)
                {
                    $Config.Eap.EapType.ServerValidation.ServerNames = $ServerNames
                }

                If ($RootCA)
                {
                    $Config.Eap.EapType.ServerValidation.TrustedRootCA = $RootCA.Replace('..','$& ')
                }
            }
            ElseIf ($Eap -eq 'TLS')
            {
                $ProfileXml = [Xml]($This.XmlTemplate(2) -f $Tag, $Hex, $Mode, $Auth, $Enc)
                $Config     = $ProfileXml.WlanProfile.Msm.Security.OneX.EapConfig.EapHostConfig.Config

                If ($ServerNames)
                {
                    $Node   = $Config.SelectNodes("//*[local-name()='ServerNames']")
                    $Node[0].InnerText = $ServerNames
                }

                If ($RootCA)
                {
                    $Node = $Config.SelectNodes("//*[local-name()='TrustedRootCA']")
                    $Node[0].InnerText = $RootCA.Replace('..','$& ')
                }
            }

            If ($Auth -eq 'WPA3SAE')
            {
                # // ------------------------------------------
                # // | Set transition mode as true for WPA3-SAE |
                # // ------------------------------------------

                $Names    = [System.Xml.XmlNamespaceManager]::New($ProfileXml.NameTable)
                $Names.AddNamespace('WLANProfile', $ProfileXml.DocumentElement.GetAttribute('xmlns'))
                $RefNode = $ProfileXml.SelectSingleNode('//WLANProfile:authEncryption', $Names)
                $XmlNode = $ProfileXml.CreateElement(
                           'transitionMode',
                           'http://www.microsoft.com/networking/WLAN/profile/v4')
                $XmlNode.InnerText = 'true'
                $Null    = $RefNode.AppendChild($XmlNode)
            }

            Return $This.FormatXml($ProfileXml.OuterXml)
        }
        Catch
        {
            Throw "Failed to create a new profile"
        }
    }
```

```
        [Object] NewWiFiProfilePsk([String]$Tag,[String]$Pass,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlPsk($Tag,'Auto','WPA2PSK','AES',$Pass)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWiFiProfilePsk([String]$Tag,[String]$Pass,[String]$Mode,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlPsk($Tag,$Mode,'WPA2PSK',"AES")
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWiFiProfilePsk([String]$Tag,[String]$Pass,[String]$Mode,[String]$Auth,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlPsk($Tag,$Mode,$Auth,'AES',$Name)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWiFiProfilePsk([String]$Tag,[String]$Pass,[String]$Mode,[String]$Auth,[String]$Enc,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlPsk($Tag,$Mode,$Auth,$Enc,$Name)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWifiProfileEap([String]$Tag,[String]$EAP,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlEap($Tag,'Auto','WPA2PSK','AES',$EAP,'',$Null)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWifiProfileEap([String]$Tag,[String]$Mode,[String]$EAP,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlEap($Tag,$Mode,'WPA2PSK','AES',$EAP,'',$Null)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWifiProfileEap([String]$Tag,[String]$Mode,[String]$Auth,[String]$EAP,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlEap($Tag,$Mode,$Auth,'AES',$EAP,'',$Null)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWifiProfileEap([String]$Tag,[String]$Mode,[String]$Auth,[String]$Enc,[String]$EAP,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlEap($Tag,$Mode,$Auth,$Enc,$EAP,'',$Null)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWifiProfileEap([String]$Tag,[String]$Mode,[String]$Auth,[String]$Enc,[String]$Eap,
        [String[]]$ServerNames,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlEap($Tag,$Mode,$Auth,$Enc,$EAP,$ServerNames,$Null)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWifiProfileEap([String]$Tag,[String]$Mode,[String]$Auth,[String]$Enc,[String]$Eap,
        [String[]]$ServerNames,[String]$RootCA,[String]$Name)
        {
            $ProfileTemp = $This.NewWifiProfileXmlEap($Tag,$Mode,$Auth,$Enc,$EAP,$ServerNames,$RootCA)
            Return $This.NewWifiProfile($ProfileTemp,$Name)
        }
        [Object] NewWifiProfileXml([String]$Type,[String]$Name,[Bool]$Overwrite)
        {
            Return $This.NewWifiProfile($Type,$Name)
        }
        NewWifiProfile([String]$Type,[String]$Name,[Bool]$Overwrite)
        {
            Try
            {
                $Guid           = $This.Adapter.Selected().Guid
                $Handle         = $This.NewWiFiHandle()
                $Flags          = 0
                $ReasonCode     = [IntPtr]::Zero
                $Ptr            = [System.Runtime.InteropServices.Marshal]::StringToHGlobalUni($Type)
                $ReturnCode     = $This.WlanSetProfile(
                                  $Handle,
                                  [Ref]$Guid,
                                  $Flags,
                                  $Ptr,
                                  [IntPtr]::Zero,
                                  $Overwrite,
```

```
                                [IntPtr]::Zero,
                                [Ref]$ReasonCode)
            $ReturnCodeMsg = $This.Win32Exception($ReturnCode)
            $ReasonCodeMsg = $This.WiFiReasonCode($ReasonCode)

            If ($ReturnCode -eq 0)
            {
                Write-Verbose -Message $ReturnCodeMsg
            }
            Else
            {
                Throw $ReturnCodeMsg
            }

            Write-Verbose -Message $ReasonCodeMsg
        }
        Catch
        {
            Throw "Failed to create the profile"
        }
        Finally
        {
            If ($Handle)
            {
                $This.RemoveWiFiHandle($Handle)
            }
        }
    }
    RemoveWifiProfile([String]$Tag)
    {
        $Handle   = $This.NewWiFiHandle()
        $xAdapter = $This.Adapter.Selected()
        (New-Object WiFi.ProfileManagement)::WlanDeleteProfile(
            $Handle,
            [Ref]$xAdapter.Guid,
            $Tag,
            [IntPtr]::Zero)
        $This.RemoveWifiHandle($Handle)
    }
    StageXamlEvent()
    {
        If ($This.Mode -ne 1)
        {
            Throw "Invalid mode"
        }

        # // ------------------
        # // | Event Triggers |
        # // ------------------

        $Wifi         = $This

        $This.Xaml.IO.Adapter.Add_SelectionChanged(
        {
            $Index    = $Wifi.Xaml.IO.Adapter.SelectedItem.Index
            If ($Index -gt -1)
            {
                $Wifi.SelectAdapter($Index)
            }
        })

        $This.Xaml.IO.Refresh.Add_Click(
        {
            $Wifi.Refresh()
            If ($Wifi.Xaml.IO.FilterText.Text -ne "")
            {
                $Wifi.SearchFilter()
            }
        })

        $This.Xaml.IO.FilterText.Add_TextChanged(
        {
```

```
        If ($Wifi.Xaml.IO.Network.Count -gt 0)
        {
            $Wifi.SearchFilter()
        }
    })

    $This.Xaml.IO.Profile.Add_SelectionChanged(
    {
        $Wifi.Xaml.IO.ProfileExtension.IsEnabled = $Wifi.Xaml.IO.Profile.SelectedIndex -ne -1
        If ($Wifi.Xaml.IO.Profile.SelectedIndex -ne -1)
        {
            $Wifi.Xaml.IO.ProfileExtension.Items.Clear()
            $Wifi.Xaml.IO.Profile.SelectedItem.Profile() | % {

                $Wifi.Xaml.IO.ProfileExtension.Items.Add($_)
            }
        }
    })

    $This.Xaml.IO.Network.Add_SelectionChanged(
    {
        $Index        = $Wifi.Xaml.IO.Network.SelectedItem.Index
        If ($Index -ne -1)
        {
            $Wifi.SelectNetwork($Index)
        }
    })

    $This.Xaml.IO.Connect.Add_Click(
    {
        If (!$Wifi.Connected -and $Wifi.Xaml.IO.Network.SelectedIndex -ne -1)
        {
            # // ----------------------------------
            # // | Establish restoration criteria |
            # // ----------------------------------

            $Target   = $Wifi.Xaml.IO.Network.SelectedItem
            $xAdapter = $Wifi.Adapter.Selected()
            $Guid     = $xAdapter.Guid
            $Count    = 0
            $State    = $Null

            # // --------------------------------
            # // | Check if the profile exists |
            # // --------------------------------

            If ($Target.Name -in $xAdapter.Profile.Output.ProfileName)
            {
                $Wifi.Connect($Target)
                Do
                {
                    $State         = $Wifi.NetshShowInterface($Wifi.Adapter.Selected().Name)
                    If ($State.State -ne "Connected")
                    {
                        Start-Sleep 1
                        $Count ++
                    }
                }
                Until ($Wifi.Connected -or $Count -gt 5)

                Switch ($Wifi.Connected)
                {
                    $False
                    {
                        [System.Windows.MessageBox]::Show("Unable to connect","Error")
                    }
                    $True
                    {
                        $Wifi.RefreshWirelessAdapterList()
                        $Adapter       = $Wifi.Adapters | ? Guid -eq $Guid
                        $Wifi.UnselectAdapter()
                        $Wifi.SelectAdapter($Adapter)
```

```
                        }
                    }
                }
                Else
                {
                    $Wifi.Passphrase($Target)
                }
            }
        })

        $This.Xaml.IO.Disconnect.Add_Click(
        {
            $Wifi.Disconnect()
            If ($Wifi.Connected)
            {
                # // ---------------------------------
                # // | Establish restoration criteria |
                # // ---------------------------------

                $Adapter       = $Wifi.Selected
                $Guid          = $Wifi.Selected.Guid
                $Count         = 0
                $State         = $Null
                $Wifi.Disconnect()

                Do
                {
                    $State         = $Wifi.NetshShowInterface($Adapter.Name)
                    If ($State.State -ne "Disconnected")
                    {
                        Start-Sleep 1
                        $Count ++
                    }
                }
                Until ($Wifi.Connected -or $Count -gt 5)

                $Wifi.RefreshWirelessAdapterList()
                $Adapter       = $Wifi.Adapters | ? Guid -eq $Guid
                $Wifi.UnselectAdapter()
                $Wifi.SelectAdapter($Adapter)
                $Wifi.Refresh()
            }
            If (!$Wifi.Connect)
            {
                $Wifi.Xaml.IO.Disconnect.IsEnabled = 0
            }
        })

        $This.Xaml.IO.Cancel.Add_Click(
        {
            $Wifi.Xaml.IO.DialogResult = $False
        })
    }
    Scan()
    {
        $This.List = @( )
        $This.Network.Clear()
        [Void][Windows.Devices.WiFi.WiFiAdapter, Windows.System.Devices, ContentType=WindowsRuntime]
        $This.List = $This.RadioFindAllAdaptersAsync()
        $This.List.Wait(-1) > $Null

        $Array     = @($This.List.Result.NetworkReport.AvailableNetworks | Sort-Object -Descending SignalBars)
        $Ct        = $Array.Count

        Switch ($This.Mode)
        {
            0 { Write-Progress -Activity Scanning -Status Scanning -PercentComplete 0 }
            1 { $This.Xaml.IO.Progress.Value = 0 }
        }

        ForEach ($Network in $Array)
        {
```

```
            $This.Network.Add($This.GetSsid($This.Network.List.Count,$Network))
            $Status              = "($($This.Network.List.Count)/$($Ct-1)"
            $Percent             =  [Long]($This.Network.List.Count * 100 / $Ct)

            Switch ($This.Mode)
            {
                0 { Write-Progress -Activity Scanning -Status $Status -PercentComplete $Percent }
                1 { $This.Xaml.IO.Progress.Value = $Percent }
            }
        }

        Switch ($This.Mode)
        {
            0 { Write-Progress -Activity Scanning -Status Complete -Completed }
            1 { $This.Xaml.IO.Progress.Value = 0 }
        }
    }
    [Object] GetSsid([UInt32]$Index,[Object]$Network)
    {
        $Item                    = [Ssid]::New($Index,$Network)
        $This.Ssid.Load($Item)
        Return $Item
    }
    Refresh()
    {
        # // ___________________________
        # // | Load the runtime types |
        # // ___________________________

        [Void][Windows.Devices.Radios.Radio, Windows.System.Devices, ContentType=WindowsRuntime]
        [Void][Windows.Devices.Radios.RadioAccessStatus, Windows.System.Devices, ContentType=WindowsRuntime]
        [Void][Windows.Devices.Radios.RadioState, Windows.System.Devices, ContentType=WindowsRuntime]

        # // ____________________________
        # // | Requesting Radio Access |
        # // ____________________________

        $This.Request = $This.RadioRequestAccess()
        $This.Request.Wait(-1) > $Null

        # // ______________________________________
        # // | Throw if unable to ascertain access |
        # // ______________________________________

        If ($This.Request.Result -ne "Allowed")
        {
            Throw "Unable to request radio access"
        }

        # // ___________________________________
        # // | Establish radio synchronization |
        # // ___________________________________

        $This.Radios = $This.RadioSynchronization()
        $This.Radios.Wait(-1) > $Null

        # // ________________________________________
        # // | Throw if unable to synchronize radios |
        # // ________________________________________

        If (!($This.Radios.Result | ? Kind -eq WiFi))
        {
            Throw "Unable to synchronize wireless radio(s)"
        }

        Start-Sleep -Milliseconds 150
        $This.Scan()

        If ($This.Mode -eq 1)
        {
            $This.Xaml.IO.Network.Items.Clear()
            ForEach ($Object in $This.Network.List)
```

```
            {
                $This.Xaml.IO.Network.Items.Add($Object)
            }
        }
    }
    SelectAdapter([UInt32]$Index)
    {
        # // ------------------------------------------
        # // | Select the adapter from its description |
        # // ------------------------------------------

        If ($Index -gt $This.Adapter.Total)
        {
            Throw "(Selection/Index) outside of the bounds of the array"
        }

        $This.Adapter.Select($Index)
        $xAdapter                           = $This.Adapter.Selected()
        $xAdapter.Connected                 = $This.NetshShowInterface($xAdapter.Name)
        If ($This.Mode -eq 1)
        {
            $This.Xaml.IO.Index.Text        = $xAdapter.IfIndex
            $This.Xaml.IO.Status.Text       = $xAdapter.Status
            $This.Xaml.IO.MacAddress.Text   = $xAdapter.MacAddress
            $This.Xaml.IO.Profile.Items.Clear()
            $This.Xaml.IO.ProfileExtension.Items.Clear()
            ForEach ($Item in $This.Adapter.Selected().Profile.Output)
            {
                $This.Xaml.IO.Profile.Items.Add($Item)
            }

            Switch -Regex ($xAdapter.Status)
            {
                Up
                {
                    $This.Xaml.IO.Ssid.Text            = $xAdapter.Connected.Ssid
                    $This.Xaml.IO.Bssid.Text           = $xAdapter.Connected.Bssid
                    $This.Xaml.IO.Disconnect.IsEnabled = 1
                    $This.Xaml.IO.Connect.IsEnabled    = 0
                }
                Default
                {
                    $This.Xaml.IO.Ssid.Text            = "<Not connected>"
                    $This.Xaml.IO.Bssid.Text           = "<Not connected>"
                    $This.Xaml.IO.Disconnect.IsEnabled = 0
                    $This.Xaml.IO.Connect.IsEnabled    = 0
                }
            }
        }
    }
    UnselectAdapter()
    {
        $This.Adapter.Unselect()
        $This.RefreshWirelessAdapterList()
        If ($This.Mode -eq 1)
        {
            $This.Xaml.IO.Index.Text        = $Null
            $This.Xaml.IO.Status.Text       = $Null
            $This.Xaml.IO.MacAddress.Text   = $Null
            $This.Xaml.IO.Profile.Items.Clear()
            $This.Xaml.IO.ProfileExtension.Items.Clear()
            $This.Xaml.IO.Ssid.Text         = $Null
            $This.Xaml.IO.Bssid.Text        = $Null
            $This.Xaml.IO.Disconnect.IsEnabled = 0
            $This.Xaml.IO.Connect.IsEnabled    = 0
        }
    }
    SelectNetwork([UInt32]$Index)
    {
        If ($Index -gt $This.Network.Total)
        {
            Throw "(Selection/Index) outside of the bounds of the array"
```

```
            }

            $This.Network.Select($Index)

            If ($This.Mode -eq 1)
            {
                If ($This.Adapter.Selected().Connected.Status -ne 'Up')
                {
                    $This.Xaml.IO.Connect.IsEnabled = 1
                }
            }
        }
        UnselectNetwork()
        {
            $This.Network.Unselect()
        }
        Disconnect()
        {
            If ($This.Adapter.Index -eq -1)
            {
                Throw "Adapter not selected"
            }

            $Index    = $This.Adapter.Index
            $xAdapter = $This.Adapter.Selected()
            $xSSID    = $This.NetshShowInterface($xAdapter.Name)

            If ($This.Adapter.Selected().State -eq "CONNECTED")
            {
                $Handle           = $This.NewWiFiHandle()
                (New-Object WiFi.ProfileManagement)::WlanDisconnect(
                $Handle,
                [Ref]$xAdapter.Guid,
                [IntPtr]::Zero)
                $This.RemoveWifiHandle($Handle)

                $This.ShowToast("Disconnected: ($($xSSID.SSID)/$($xSSID.BSSID))")
                $This.Connected   = $Null

                $This.Adapter.Unselect()
                $This.Adapter.Select($Index)
            }
        }
        Connect([Object]$Target)
        {
            $Index          = $This.Adapter.Selected().Index
            $This.Adapter.Unselect()
            $This.Adapter.Select($Index)

            If ($Target.Name -in $This.Adapter.Selected().Profile.Output.ProfileName)
            {
                $Param  = $This.GetWiFiConnectionParameter($Target.Name)
                $Handle = $This.NewWiFiHandle()
                (New-Object WiFi.ProfileManagement)::WlanConnect(
                $Handle,
                [Ref]$This.Adapter.Selected().Guid,
                [Ref]$Param,
                [IntPtr]::Zero)
                $This.RemoveWifiHandle($Handle)

                $This.UnselectAdapter()
                $This.SelectAdapter($Index)

                $This.ShowToast("Connected: $($Target.Name)")
                $This.Connected = $This.NetshShowInterface($This.Adapter.Selected().Name)
            }
            If ($Target.Name -notin $This.Adapter.Selected().Profile.Output.ProfileName)
            {
                If ($Target.Authentication -match "PSK")
                {
                    $This.Passphrase($Target)
                    $This.Connected = $This.NetshShowInterface($This.Adapter.Selected().Name)
```

```
            }
            Else
            {
                Write-Host "Eas/Peap not yet implemented"
            }
        }
    }
    ShowToast([String]$Message)
    {
        $Splat = @{

            Message = $Message
            Header  = [DateTime]::Now
            Body    = $Message
            Image   = $This.OEMLogo
        }

        Show-ToastNotification @Splat
    }
    Passphrase([Object]$Target)
    {
        $Index     = $This.Selected.Index
        $xAdapter  = $This.Selected
        $Auth      = $Null
        $Enc       = $Null

        If ($Target.Authentication -match "RsnaPsk")
        {
            $Auth  = "WPA2PSK"
        }
        If ($Target.Encryption -match "Ccmp")
        {
            $Enc   = "AES"
        }

        # // ___________________________________________________________________
        # // | Passphrase collection when using the command line interface... |
        # // -------------------------------------------------------------------

        If ($This.Mode -eq 0)
        {
            $PW     = Read-Host -AsSecureString -Prompt "Enter passphrase for Network: [$($Target.SSID)]"

            $ProfileXml = $This.NewWifiProfileXmlPsk($Target.Name,"Manual",$Auth,$Enc,$PW)
            $This.NewWifiProfile($ProfileXml,$This.Selected.Name,$True)

            $Param  = $This.GetWiFiConnectionParameter($Target.Name)
            $Handle = $This.NewWiFiHandle()
            $This.WlanConnect($Handle,[Ref]$xAdapter.Guid,[Ref]$Param,[IntPtr]::Zero)
            $This.RemoveWifiHandle($Handle)

            Start-Sleep 3
            $This.UnselectAdapter()
            $This.SelectAdapter($Index)

            Switch ([UInt32]!!$This.Connected)
            {
                0
                {
                    $This.ShowToast("Connected: $($Target.Name)")
                }
                1
                {
                    $This.RemoveWifiProfile($Target.Name)
                    $This.ShowToast("Unsuccessful: Passphrase failure")
                }
            }
        }

        # // ___________________________________________________________________
        # // | Passphrase collection when using the graphical user interface... |
        # // -------------------------------------------------------------------
```

```
            If ($This.Mode -eq 1)
            {
                $Pass    = $This.GetPassphraseXaml()
                $Pass.IO.Connect.Add_Click(
                {
                    $Password      = $Pass.IO.Passphrase.Password
                    $PW            = $Password | ConvertTo-SecureString -AsPlainText -Force
                    $ProfileXml    = $This.NewWifiProfileXmlPsk($Network.Name,"manual",$Auth,$Enc,$PW)
                    $This.NewWifiProfile($ProfileXml,$Target.Name,$True)

                    $Param         = $This.GetWiFiConnectionParameter($Target.Name)
                    $Handle        = $This.NewWiFiHandle()
                    $This.WlanConnect($Handle,[Ref]$xAdapter.Guid,[Ref]$Param,[IntPtr]::Zero)
                    $This.RemoveWifiHandle($Handle)

                    Start-Sleep 3
                    $This.UnselectAdapter()
                    $This.SelectAdapter($Index)

                    If ($This.Connected)
                    {
                        $Pass.IO.DialogResult = 1
                        $This.ShowToast("Connected: $($Target.Name)")
                    }
                    If (!$This.Connected)
                    {
                        $This.RemoveWifiProfile($Target.Name)
                        $This.ShowToast("Unsuccessful: Passphrase failure")
                    }
                })

                $Pass.IO.Cancel.Add_Click(
                {
                    $Pass.IO.DialogResult = $False
                })

                $Pass.Invoke()
            }
        }
        SearchFilter()
        {
            If ($This.Xaml.IO.FilterText.Text -ne "" -and $This.Network.List.Count -gt 0)
            {
                Start-Sleep -Milliseconds 50
                $This.Xaml.IO.Network.Items.Clear()
                $Property = $This.Xaml.IO.FilterProperty.SelectedItem.Content
                $Text     = $This.Xaml.IO.FilterText.Text
                ForEach ($Item in $This.Network.List | ? $Property -match $Text)
                {
                    $This.Xaml.IO.Network.Items.Add($Item)
                }
            }
            Else
            {
                Start-Sleep -Milliseconds 50
                $This.Xaml.IO.Network.Items.Clear()
                ForEach ($Item in $This.Network.List)
                {
                    $This.Xaml.IO.Network.Items.Add($Item)
                }
            }
        }
    }
```

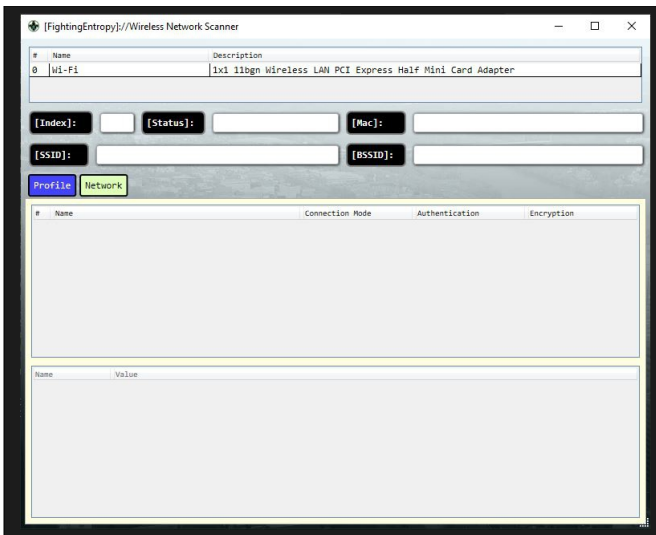
For the most part, the controller class above does very much the same thing as it did beforehand in the video.

However, the graphical user interface has been changed a fair amount, AND, it has added functionality in reference to the profiles and the information that can be seen.

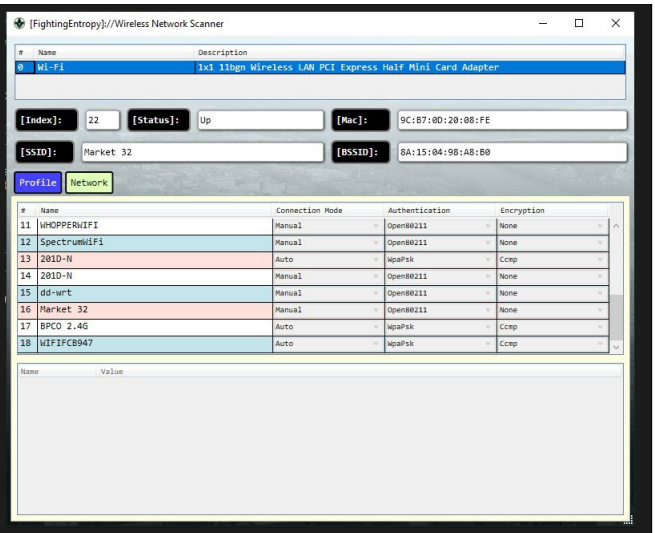
_____/ WirelessController

Images

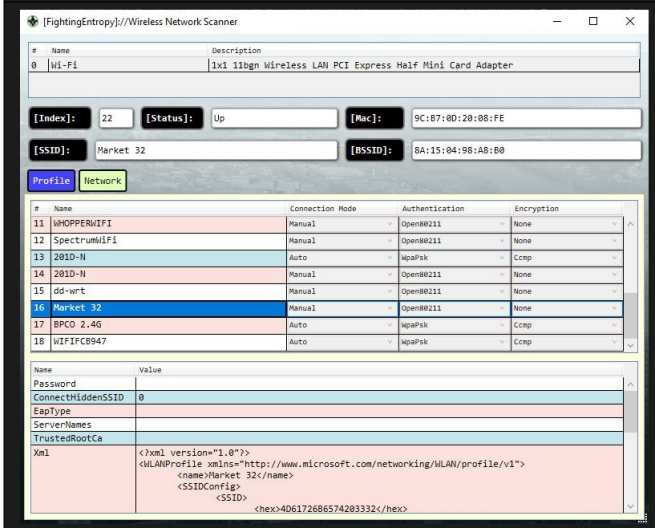
While this utility is far from completion, here are some screenshots... (Click to view larger version)



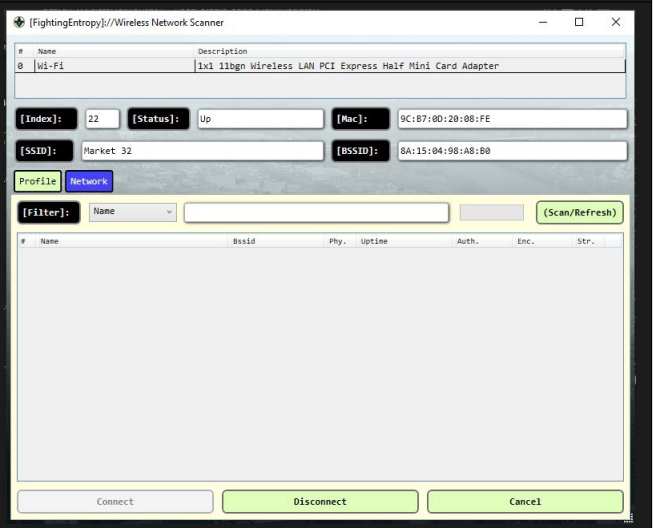
Starting position of the graphical user interface



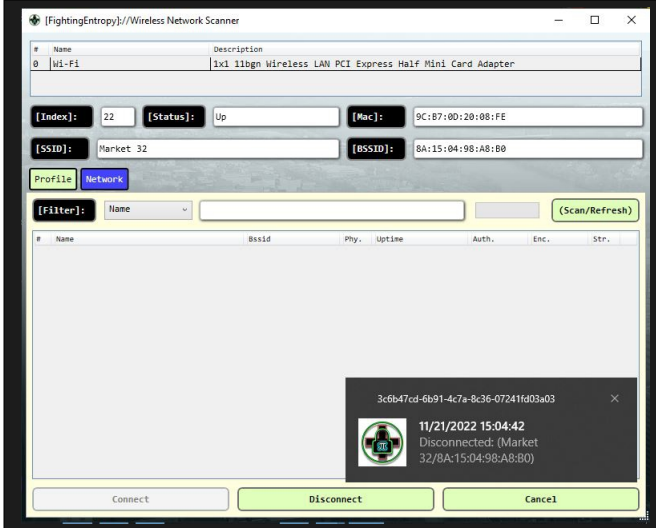
...once an adapter is selected



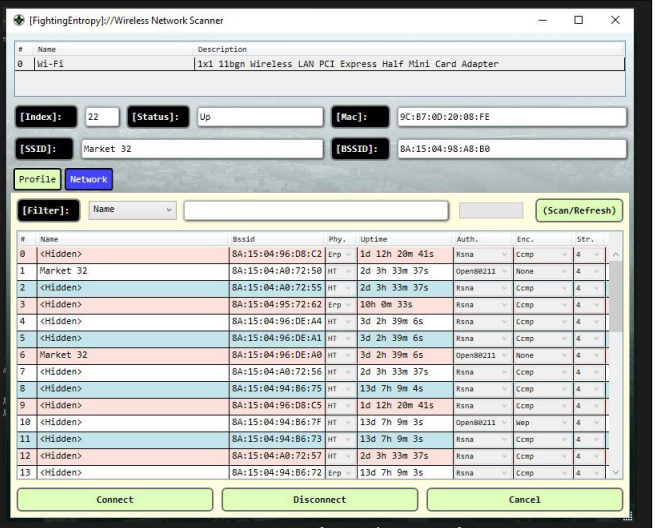
...when selecting a particular profile in the list



...Network tab when already connected to an access point



...when the network is disconnected from

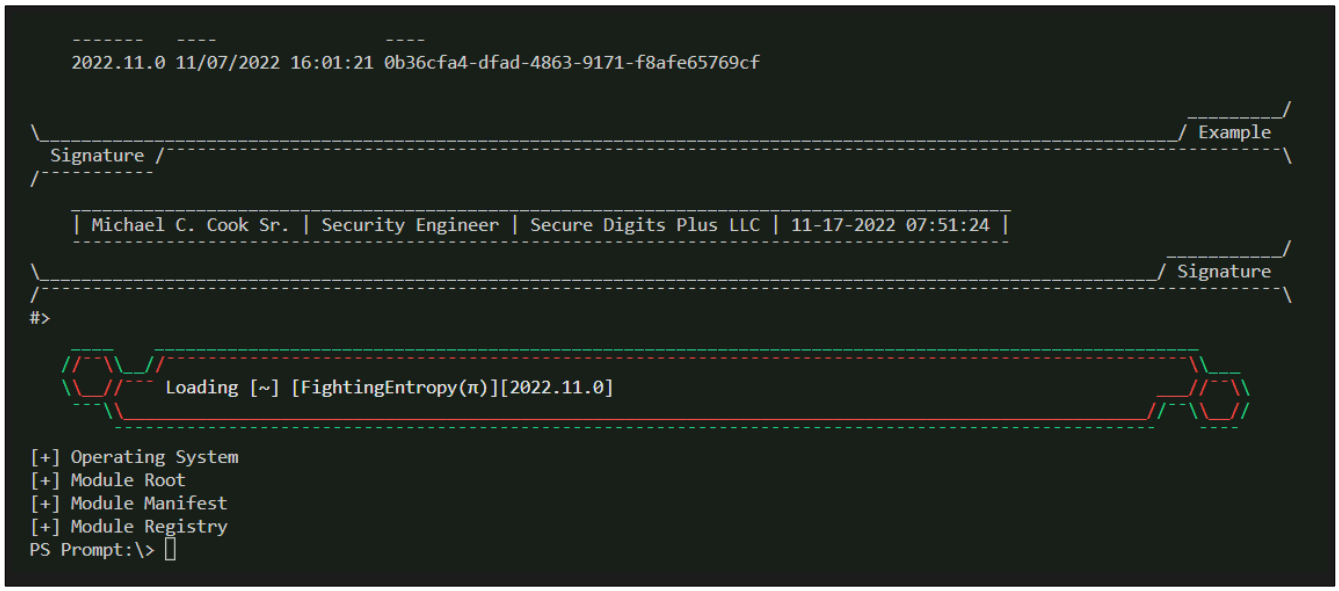


...when hitting the (Scan/Refresh) button

There are some obvious glitches occurring in some of the shots above.
However, in order to show the progress I'm making with the utility...?

I can't spend entire weeks mulling over every little decision I have to make.
I need to showcase examples of how work is being completed.

The fact of the matter is that I've been shoring up nearly every other function in the module (49)...
...everything short of New-FEInfrastructure (that's like 11191 lines of code alone, for that function).



That means I've resampled the other controls, functions, and the graphics.
Really, the things that need the most work are just the functions.
The graphics and controls are MOSTLY static assets, and don't need to be consistently updated.

So, the next order of business would be to cover the CLI (input/output) of the wireless utility.
I will cover the new version of the module relatively soon, probably in the next week or two.

Images
Output

By the way, all of those classes up above are wrapped by the following code...

```powershell
<#____  Script introduction goes here ____#>

Function Search-WirelessNetwork
{
    [CmdLetBinding()]Param([Parameter()][UInt32]$Mode)

    <#____ All of those classes above go right here ____#>

    Switch ($Mode)
    {
        0
        {
            [WirelessController]::New(0)
        }
        1
        {
            $Wifi      = [WirelessController]::New(1)
            $Wifi.Xaml.Invoke()
        }
    }
}
```

I'll cover mode 0, which is the CLI mode. Mode 1 is the GUI mode.

```
PS Prompt:\> $Wifi = [WirelessController]::New(0)
PS Prompt:\> $Wifi

Adapter               Network               Connected
-------               -------               ---------
WirelessSubcontroller WirelessSubcontroller


PS Prompt:\>
```

So, the “connected” property is going to be totally waxed.
It is there for a legacy purpose as I transition from the prior methodology I was using, for a newer one.

Over the last 2+ weeks, I've made so many design changes because I want the module to be as QUICK as possible, whenever someone loads it into memory. Having various dependencies means that someone's gotta download and install THAT one, and then THIS one, and so on and so forth.

As can be seen in the `[WirelessController]` class, the command “Get-FEModule -Mode 1” is clearly visible.
Here's what that does.

```
PS Prompt:\> $Module = Get-FEModule -Mode 1
PS Prompt:\> $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
Version     : 2022.11.0
OS          : <FightingEntropy.Module.OS>
Root        : <FightingEntropy.Module.Root>
Manifest    : <FightingEntropy.Module.Manifest>
Registry    : <FightingEntropy.Module.RegistryKey>
Manifest    : <FightingEntropy.Module.Manifest>
Registry    : <FightingEntropy.Module.RegistryKey>

PS Prompt:\>
```

All of these things are accessible from that single command.
This is the point of the module, is to act as an entry point into various other files, classes, methods, functions, and et cetera.

There are plenty of programs out there that already do this.
Pretty sure that the `[Deployment and Imaging Service Module]` by `[Microsoft]` does this, with COM objects.

The properties in the above box are important for reconstituting the various resources needed by the module.
In this particular circumstance, there are (2) things needed from the module, `[FightingEntropy(π)]`.

The first is the `"OEMLogo.bmp"` graphic for the toast notification (screenshot 005 in the earlier section)
The second is the Type Definition within the file `"Wifi.cs"` saved in the control path.

To access the OEMLogo.bmp file, `$This.Module._Graphic("OEMLogo.bmp").Fullname`
To access the Wifi.cs file, `Add-Type -Path $This.Module._Control("Wifi.cs").Fullname -ErrorAction Continue`

There are also the graphics which are statically linked within the Xaml classes, `"icon.ico"`, and `"OEMbg.jpg"`.
However, since those are statically linked, there's no need to write them into the class at all.

Lets expand the properties for `$Wifi.Adapter`

```
PS Prompt:\> $Wifi.Adapter

Type    List    Total Index
----    ----    ----- -----
Adapter {Wi-Fi}     1    -1


PS Prompt:\>
```

In this particular property, the “List” property contains all of the available wireless adapters that were detected by the command Get-NetAdapter...

```
    $This.Adapter.Clear()
    ForEach ($Adapter in Get-NetAdapter | ? PhysicalMediaType -match "(Native 802.11|Wireless (W|L)AN)")
    {
        $Item          = [WifiInterface]::New($This.Adapter.Total,$Adapter)
        $This.GetWiFiProfileList($Item)
        $This.Adapter.Add($Item)
    }
```

What that method above is actually doing, is looking through something SIMILAR to:
`Get-Ciminstance MSFT_NetAdapter -Namespace ROOT\StandardCimv2`

The command Get-NetAdapter is NOT exactly the same, because for some reason it doesn't have the PhysicalMediaType, which is really the only property I could see that would be able to filter out what is a wireless adapter, or not.

Then again, perhaps there are some other properties I could use to determine that.
However, the available wireless adapters on the system are all found via:
`Get-NetAdapter | ? PhysicalMediaType -match "(Native 802.11|Wireless (W|L)AN)"`

If I access that property `$Wifi.Adapter.List`, here's what I will see...

```
PS Prompt:\> $Wifi.Adapter.List

Index       : 0
Name        : Wi-Fi
Guid        : e3a47a46-9920-469e-996c-422138000d09
Description : 1x1 11bgn Wireless LAN PCI Express Half Mini Card Adapter
IfIndex     : 22
Status      : Disconnected
MacAddress  : 9C:B7:0D:20:08:FE
LinkSpeed   : 72 Mbps
State       : Disconnected
Profile     : WifiProfileList
```

Here, there is a single object showing. If there were multiple wireless adapters in this system, it would show the others as well. It is pulling a lot of information while also injecting the “Profile” property.

Right now, that adapter isn't selected. To select it, we use `$Wifi.Adapter.Select(0)`.
The 0 is the index. This is NOT the same thing as the InterfaceIndex, which THAT property, is “IfIndex”.
Once `$Wifi.Adapter.Select(0)` is processed, then the method `$Wifi.Adapter.Selected()` will return that adapter.

Previously, I had a method that was able to select it from the list, and make amendments to that object.
However, rather than to point to the original object whenever changes were made, it would duplicate the object.

Part of instantiation, is that a New-Object is being created at that time, rather than an existing one being duplicated or amended. Being able to get your head around THAT particular fact, takes a while of working with complicated objects for a while, before you start to understand what the code is doing between memory and the CPU, or the operating system and `[PowerShell]` host.

NOW, what I'd like to do, is to look at the profiles under `$Wifi.Adapter.Selected().Profile`

```
PS Prompt:\> $Wifi.Adapter.Selected().Profile

Interface     Process                                                  Output
---------     -------                                                  ------
WifiInterface {TOH Public 2.4Ghz, Subway, Uncommon Grounds, Courtyard_Guest...} {Wi-Fi, Wi-Fi, Wi-Fi...}
```

Here, we can see the properties “Interface”, “Process”, and “Output”, this is the `[WifiProfileList]` object.
This process is actually rather convoluted, however, the result is efficiency... so I'll explain.

When the adapters are polled with `Get-NetAdapter | ? PhysicalMediaType -match "(Native 802.11|Wireless (W|L)AN)"` in the method above, what happens is that the interface throws itself into a `[WifiProfileList]` object, where ALL of the profiles for that specific adapter are populated and placed into the “Process” property.

Then, the main `[WirelessController]` class has a property that has the `[WifiProfileSubcontroller]` object, which that has properties that refer to it's various `[AuthenticationList]`, `[EncryptionList]`, and `[ConnectionModeList]` objects, and the method `$This.GetWiFiProfileList($Item)` is what does this.

Each of these things are populated with their corresponding Enum types, and slot objects, and when the profile is thrown into the Load() method in the `[WifiProfileSubcontroller]`, it will retrieve all of that information in a template, and then return that template in the property “Output”.

It isn't exactly INTUITIVE, however- if I want the function to be (incredibly lightweight/ultra responsive), not need a ton of (modules/resources), and pack a powerful punch...? Uh, this is probably the way to do it.

It's like, imagine 50 guys were standing around chuckling like a bunch of kindergartners, but then- uh oh. Bruce Lee literally showed up with his game face on. When 50 guys see Bruce Lee with his game face on...? ...they're not gonna be chuckling like a bunch of kindergartners... not anymore, dude. No way...

```
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
```

```
Guy[1] : Uh- ...the guy that just showed up just now, that's Bruce Lee, isn't it...?
Guy[2] : *nervous gulp* Yeh.
Guy[3] : I'm outta here, dude... I'm not goin' up against this tough as nails, psycho...
```

```
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
```

So, here's what happens when I call `$Wifi.Adapter.Selected().Profile.Process`

```
PS Prompt:\> $Wifi.Adapter.Selected().Profile.Process

Index Name                 Flags   Detail
----- ----                 -----   ------
    0 TOH Public 2.4Ghz    AllUser Wifi.ProfileManagement+ProfileInfo
    1 Subway               AllUser Wifi.ProfileManagement+ProfileInfo
    2 Uncommon Grounds     AllUser Wifi.ProfileManagement+ProfileInfo
    3 Courtyard_Guest      AllUser Wifi.ProfileManagement+ProfileInfo
    4 library              AllUser Wifi.ProfileManagement+ProfileInfo
    5 Uncommon Grounds 2.4 AllUser Wifi.ProfileManagement+ProfileInfo
    6 MySpectrumWiFi8F-2G  AllUser Wifi.ProfileManagement+ProfileInfo
    7 Library Wireless     AllUser Wifi.ProfileManagement+ProfileInfo
    8 Doug99999            AllUser Wifi.ProfileManagement+ProfileInfo
    9 redroof              AllUser Wifi.ProfileManagement+ProfileInfo
   10 It Hurts When IP     AllUser Wifi.ProfileManagement+ProfileInfo
   11 WHOPPERWIFI          AllUser Wifi.ProfileManagement+ProfileInfo
   12 SpectrumWiFi         AllUser Wifi.ProfileManagement+ProfileInfo
   13 201D-N 2             AllUser Wifi.ProfileManagement+ProfileInfo
   14 201D-N               AllUser Wifi.ProfileManagement+ProfileInfo
   15 dd-wrt               AllUser Wifi.ProfileManagement+ProfileInfo
   16 Market 32            AllUser Wifi.ProfileManagement+ProfileInfo
   17 BPCO 2.4G            AllUser Wifi.ProfileManagement+ProfileInfo
   18 WIFIFCB947           AllUser Wifi.ProfileManagement+ProfileInfo

PS Prompt:\>
```

If I want to expand the properties for the class that jcwalker wrote in `[C#]`, I can use:
`$Wifi.Adapter.Selected().Profile.Process.Detail | Format-Table`

```
PS Prompt:\> $Wifi.Adapter.Selected().Profile.Process.Detail | Format-Table

ProfileName          ConnectionMode Authentication Encryption Password ConnectHiddenSSID EAPType ServerNames
-----------          -------------- -------------- ---------- -------- ----------------- ------- -----------
TOH Public 2.4Ghz    manual         Open80211      none                              False
Subway               manual         Open80211      none                              False
Uncommon Grounds     manual         Open80211      none                              False
Courtyard_Guest      manual         Open80211      none                              False
library              manual         Open80211      none                              False
Uncommon Grounds 2.4 manual         Open80211      none                              False
MySpectrumWiFi8F-2G  manual         WpaPsk         Ccmp                              False
Library Wireless     manual         Open80211      none                              False
Doug99999            manual         Open80211      none                              False
redroof              manual         Open80211      none                              False
It Hurts When IP     auto           WpaPsk         Ccmp                              False
WHOPPERWIFI          manual         Open80211      none                              False
SpectrumWiFi         manual         Open80211      none                              False
```

```
201D-N              auto          WpaPsk        Ccmp                        False
201D-N              manual        Open80211     none                        False
dd-wrt              manual        Open80211     none                        False
Market 32           manual        Open80211     none                        False
BPCO 2.4G           auto          WpaPsk        Ccmp                        False
WIFIFCB947          auto          WpaPsk        Ccmp                        False

PS Prompt:\>
```

Now, this doesn't have the information that is in the property “Output” (and a couple of properties don't fit). It's worth noting that if I use something like `$Wifi.Adapter.Selected().Profile.Process[0].Detail.Authentication` it comes back with the string of “Open80211”.

That doesn't come with the index, or the description.
However, `$Wifi.Adapter.Selected().Profile.Output | Format-Table`

```
PS Prompt:\>
PS Prompt:\> $Wifi.Adapter.Selected().Profile.Output | Format-Table

Index ProfileName          ConnectionMode Authentication Encryption Password ConnectHiddenSSID EapType
----- -----------          -------------- -------------- ---------- -------- ----------------- -------
    0 TOH Public 2.4Ghz    Manual         Open80211      None                              0         {}
    1 Subway               Manual         Open80211      None                              0         {}
    2 Uncommon Grounds     Manual         Open80211      None                              0         {}
    3 Courtyard_Guest      Manual         Open80211      None                              0         {}
    4 library              Manual         Open80211      None                              0         {}
    5 Uncommon Grounds 2.4 Manual         Open80211      None                              0         {}
    6 MySpectrumWiFi8F-2G  Manual         WpaPsk         Ccmp                              0         {}
    7 Library Wireless     Manual         Open80211      None                              0         {}
    8 Doug99999            Manual         Open80211      None                              0         {}
    9 redroof              Manual         Open80211      None                              0         {}
   10 It Hurts When IP     Auto           WpaPsk         Ccmp                              0         {}
   11 WHOPPERWIFI          Manual         Open80211      None                              0         {}
   12 SpectrumWiFi         Manual         Open80211      None                              0         {}
   13 201D-N               Auto           WpaPsk         Ccmp                              0         {}
   14 201D-N               Manual         Open80211      None                              0         {}
   15 dd-wrt               Manual         Open80211      None                              0         {}
   16 Market 32            Manual         Open80211      None                              0         {}
   17 BPCO 2.4G            Auto           WpaPsk         Ccmp                              0         {}
   18 WIFIFCB947           Auto           WpaPsk         Ccmp                              0         {}

PS Prompt:\>
```

If I go to use `$Wifi.Adapter.Selected().Profile.Output[0].Authentication`, I'll get this back...

```
PS Prompt:\> $Wifi.Adapter.Selected().Profile.Output[0].Authentication

Index Type       Description
----- ----       -----------
    2 Open80211 {Open authentication over 802.11 wireless., Devices are authenticated and can connect...

PS Prompt:\>
```

There are various other changes that I've had to go back and make to this particular class, as I was writing this document. For instance, I added a method that actually shows the hidden properties for the adapter.

`[PowerShell]` has some caveats about it where the hidden properties are sorta like private properties, though they're not identical to each other. Hidden properties are simply hidden, they can still be accessed by typing them explicitly. Regardless, here is what I've added.

```
PS Prompt:\> $Wifi.Adapter.Selected().Profile.Output[0] | Select Index, Name, Guid, Description, IfIndex,
Status, MacAddress, LinkSpeed, State, ProfileName, ConnectionMode, Authentication, Encryption, Password,
ConnectHiddenSSID, EapType, ServerNames, TrustedRootCA, Xml

Index             : 0
Name              : Wi-Fi
Guid              : e3a47a46-9920-469e-996c-422138000d09
Description       : 1x1 11bgn Wireless LAN PCI Express Half Mini Card Adapter
IfIndex           : 22
Status            : Disconnected
```

```
MacAddress        : 9C:B7:0D:20:08:FE
LinkSpeed         : 72 Mbps
State             : Disconnected
ProfileName       : TOH Public 2.4Ghz
ConnectionMode    : Manual
Authentication    : Open80211
Encryption        : None
Password          :
ConnectHiddenSSID : 0
EapType           :
ServerNames       : {}
TrustedRootCA     :
Xml               : <?xml version="1.0"?>
                    <WLANProfile xmlns="http://www.microsoft.com/networking/WLAN/profile/v1">
                        <name>TOH Public 2.4Ghz</name>
                        <SSIDConfig>
                                <SSID>
                                        <hex>544F48205075626C696320322E3447687A</hex>
                                        <name>TOH Public 2.4Ghz</name>
                                </SSID>
                        </SSIDConfig>
                        <connectionType>ESS</connectionType>
                        <connectionMode>manual</connectionMode>
                        <MSM>
                                <security>
                                        <authEncryption>
                                                <authentication>open</authentication>
                                                <encryption>none</encryption>
                                                <useOneX>false</useOneX>
                                        </authEncryption>
                                </security>
                        </MSM>
                        <MacRandomization xmlns="http://www.microsoft.com/networking/WLAN/profile/v3">
                                <enableRandomization>false</enableRandomization>
                        </MacRandomization>
                    </WLANProfile>

PS Prompt:\>
```

This looks incredibly similar to the output from netsh, actually. Though to be clear, there's a lot more here.

If the adapter is CURRENTLY connected to an access point, that's going to affect the GUI somehow.
This is sorta what I've been having to mull over, actually. As for the profile portion of the (adapter/interface) object, let's backpedal to the main property of the controller class.

```
PS Prompt:\> $Wifi

Adapter               Network               Connected
-------               -------               ---------
WirelessSubcontroller WirelessSubcontroller


PS Prompt:\> $Wifi.Network

Type    List Total Index
----    ---- ----- -----
Network {}       0    -1

PS Prompt:\>
```

So, as we can see here, there's nothing in the network list. If I use the method `$Wifi.Refresh()`...

```
PS Prompt:\> $Wifi.Refresh()
PS Prompt:\> $Wifi.Network

Type    List                                        Total Index
----    ----                                        ----- -----
Network {<Hidden>, <Hidden>, <Hidden>, Market 32...}    32    -1

PS Prompt:\>
```

If I expand the property for the list object like so... `$Wifi.Network.List | Select-Object Index, Name, Bssid, Physical, Network, Uptime, Authentication, Encryption, Strength | Format-Table`

```
Index Name                     Bssid             Physical        Network Uptime         Authentication Encryption Strength
----- ----                     -----             --------        ------- ------         -------------- ---------- --------
    0 <Hidden>                 8A:15:04:A0:72:5F HT       Infrastructure 2d 6h 16m 8s   Open80211      Wep              4
    1 <Hidden>                 8A:15:04:A0:72:52 Erp      Infrastructure 2d 6h 16m 7s   Rsna           Ccmp             4
    2 <Hidden>                 8A:15:04:A0:72:56 HT       Infrastructure 2d 6h 16m 7s   Rsna           Ccmp             4
    3 Market 32                8A:15:04:96:DE:A0 HT       Infrastructure 3d 5h 21m 37s  Open80211      None             4
    4 <Hidden>                 8A:15:04:96:DE:A1 HT       Infrastructure 3d 5h 21m 37s  Rsna           Ccmp             4
    5 <Hidden>                 8A:15:04:96:DE:A3 HT       Infrastructure 3d 5h 21m 37s  Rsna           Ccmp             4
    6 <Hidden>                 8A:15:04:95:72:6F HT       Infrastructure 12h 43m 5s     Open80211      Wep              4
    7 <Hidden>                 8A:15:04:94:B6:71 HT       Infrastructure 13d 9h 51m 34s Rsna           Ccmp             4
    8 <Hidden>                 8A:15:04:94:B6:77 HT       Infrastructure 13d 9h 51m 34s Rsna           Ccmp             4
    9 <Hidden>                 8A:15:04:94:B6:74 HT       Infrastructure 13d 9h 51m 34s Rsna           Ccmp             4
   10 <Hidden>                 8A:15:04:95:72:66 HT       Infrastructure 12h 43m 4s     Rsna           Ccmp             4
   11 <Hidden>                 8A:15:04:96:D8:C5 HT       Infrastructure 1d 15h 3m 12s  Rsna           Ccmp             4
   12 <Hidden>                 8A:15:04:94:B6:75 HT       Infrastructure 13d 9h 51m 34s Rsna           Ccmp             4
   13 <Hidden>                 8A:15:04:96:DE:A6 HT       Infrastructure 3d 5h 21m 37s  Rsna           Ccmp             4
   14 <Hidden>                 8A:15:04:98:A8:B6 HT       Infrastructure 2h 47m 25s     Rsna           Ccmp             4
   15 <Hidden>                 8A:15:04:98:A8:B5 HT       Infrastructure 2h 47m 25s     Rsna           Ccmp             4
   16 <Hidden>                 8A:15:04:98:A8:B7 HT       Infrastructure 2h 47m 25s     Rsna           Ccmp             4
   17 <Hidden>                 8A:15:04:98:A8:B3 HT       Infrastructure 2h 47m 25s     Rsna           Ccmp             4
   18 <Hidden>                 8A:15:04:98:A8:B4 HT       Infrastructure 2h 47m 25s     Rsna           Ccmp             4
   19 Market 32                8A:15:04:98:A8:B0 HT       Infrastructure 2h 47m 24s     Open80211      None             4
   20 <Hidden>                 8A:15:04:98:A8:B2 Erp      Infrastructure 2h 47m 25s     Rsna           Ccmp             4
   21 <Hidden>                 8A:15:04:96:DE:A7 HT       Infrastructure 3d 5h 21m 37s  Rsna           Ccmp             4
   22 <Hidden>                 8A:15:04:96:DE:A5 HT       Infrastructure 3d 5h 21m 37s  Rsna           Ccmp             4
   23 <Hidden>                 8A:15:04:98:A8:B1 HT       Infrastructure 2h 47m 25s     Rsna           Ccmp             4
   24 <Hidden>                 8A:15:04:98:A8:BF HT       Infrastructure 2h 47m 24s     Open80211      Wep              4
   25 <Hidden>                 8A:15:04:96:DE:A2 Erp      Infrastructure 3d 5h 21m 37s  Rsna           Ccmp             4
   26 <Hidden>                 8A:15:04:96:DE:A4 HT       Infrastructure 3d 5h 21m 37s  Rsna           Ccmp             4
   27 Harvest Grain Pizza      B4:2A:0E:8F:91:C3 HT       Infrastructure 4h 50m 23s     RsnaPsk        Ccmp             3
   28 Marcella                 CC:88:C7:8F:D5:C2 HT       Infrastructure 1d 14h 40m 50s RsnaPsk        Ccmp             3
   29 <Hidden>                 00:1D:D4:9B:7E:00 HT       Infrastructure 4d 2h 43m 59s  RsnaPsk        Ccmp             2
   30 ParklaneTobacconist      18:E8:29:97:24:23 HT       Infrastructure 19d 16h 0m 24s RsnaPsk        Ccmp             2
   31 DIRECT-f9-HP M402 LaserJet EA:9E:B4:23:02:F9 HT     Infrastructure 4d 2h 45m 23s  RsnaPsk        Ccmp             2


PS Prompt:\>
```

As was the same with the profile properties for the adapter, I can expand certain properties to withdraw objects... `$Wifi.Network.List[0].Physical`

```
PS Prompt:\> $Wifi.Network.List[0].Physical

Index Type Description
----- ---- -----------
    7 HT   {(HT/High Throughput [802.11n])}

PS Prompt:\>
```

`$Wifi.Network.List[0].Authentication`

```
PS Prompt:\> $Wifi.Network.List[0].Authentication

Index Type      Description
----- ----      -----------
    2 Open80211 {Open authentication over 802.11 wireless., Devices are authenticated and can connect...

PS Prompt:\>
```

`$Wifi.Network.List[0].Encryption`

```
PS Prompt:\> $Wifi.Network.List[0].Encryption

Index Type Description
----- ---- -----------
    2 Wep  {Specifies a WEP cipher algorithm with a cipher key of any length.}


PS Prompt:\>
```

_____/
_____/ Output

Conclusion

Alright, so, there was a lot of stuff covered in this document.
First of all, this function is the LAST THING that I need to finish before I fully update the module version.
Well, New-FEInfrastructure needs some updates as well.

I may very well hold off until I have the time to go through those functions...

[2022.10.1], [2022.11.0] are both sorta (dummy/template) versions...

I've been toying around with the idea for like the last year or so, to fully implement the stuff I've been adding to my Github project that wouldn't exactly hurt to have alongside the rest of the components that I've developed.

The one function that I'd REALLY like to add to the module is the EventLog-Utility.
Something that takes a fair amount of time to collect every event log on a system, and then export to a zip file.

Regardless, that about does it for this particular document.

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