```
<#
     ____    ____________________________________________________________________________________________________
    //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
    \\__//¯¯¯ [FightingEntropy(π)][2023.4.0]: 2023-04-03 18:55:27                                        ___//¯¯\\
     ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
         ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯

\________________________________________________________________________________________________________________________/
  About /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯

    --------------------------------------------------------------------------------------------
    | https://github.com/mcc85s/FightingEntropy/blob/main/Version/2023.4.0/FightingEntropy.ps1 |
    --------------------------------------------------------------------------------------------

    [FightingEntropy(π)] is a modification for [Windows PowerShell] that is meant for various tasks related to:

    [+] [system administration]
    [+] [networking]
    [+] [virtualization]
    [+] [security]
    [+] [graphic design]
    [+] [system management/maintenance]

    ...it'll eventually be usable on ALL platforms where [PowerShell] is able to be deployed.

     ________________________________________________________________________________
    //¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\
    \\ Demo                                                                          //
    //______________________________________________________________________________\\
    \\¯¯¯¯¯¯¯¯¯¯||¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯||¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯//
    // Date     || Name                             || Url                          \\
    \\__________||__________________________________||______________________________//
    //¯¯¯¯¯¯¯¯¯¯||¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯||¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\
    \\ 10/28/22 || [FightingEntropy(π)][2022.10.1]  || https://youtu.be/S7k4lZdPE-I //
    // 04/03/23 || Virtualization Lab - TCP Session || https://youtu.be/09c-fFbEQrU \\
    \\__________||__________________________________||______________________________//
     ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
    This module is rather [experimental] and incorporates [a lot of moving parts],
    so it has [many areas of development].

    The [end goal] of this [module], is to provide [heightened security] and [protection] against:

    [+] [identity theft]
    [+] [cybercriminals]
    [+] [douchebags]
    [+] [malware]
    [+] [viruses]
    [+] [ransomware]
    [+] [hackers who have malicious intent]

    Many of the tools in the wild are able to be circumvented by some of these [hackers] and [cybercriminals].
    If you don't believe me...? That's fine.

    That's why this link to a particular website about a particular event, exists.
    -----------------------------------------------------------------------------------
    | https://en.wikipedia.org/wiki/2020_United_States_federal_government_data_breach |
    -----------------------------------------------------------------------------------
    Even the experts make mistakes.

    [FightingEntropy(π)] is meant to extend many of the capabilities that come with [Windows].

    This file acts as the [installation/removal] process as well as for performing [validation] and [testing] purposes.

    It is effectively a [shell] of the [entire module], and can be used to implement [updates] to the [module itself],
    in a similar manner to how (Continuous Integration/Continuous Development) works (still a work in progress).
    ---------------------------------------------------------------------------
    | [FightingEntropy(π)][2023.4.0]                                          |
    |=========================================================================|
    | Version    | Date                  | Guid                                  |
    |-----------|---------------------|-------------------------------------|
    | 2023.4.0   | 04/03/2023 18:53:49 | 75f64b43-3b02-46b1-b6a2-9e86cccf48113 |
    ---------------------------------------------------------------------------

    -----------------
    | Prerequisites |
    -----------------
    1) A system running [Windows PowerShell] on:
       - [Windows 10/11]
       - [Windows Server 2016/2019/2021]
    2) [Execution Policy] must be set to [bypass]

    3) Must be running a [PowerShell] session with [administrative privileges]

    ----------------
    | Installation |
    ----------------

    1) [Load the module into memory], which can be done be using this command:

    ----------------------------------------------------------------------------------------------
    | irm https://github.com/mcc85s/FightingEntropy/blob/main/FightingEntropy.ps1?raw=true | iex |
    ----------------------------------------------------------------------------------------------

    ...or just (copying + pasting) the content of the file...

    https://github.com/mcc85s/FightingEntropy/blob/main/FightingEntropy.ps1

    ...into the [PowerShell] session, and pressing <enter>

    2) Once the [module is loaded into memory], enter the following:

    ----------------------------------
    | Operation  | Instructions      |
    |============|===================|
    | Install    | $Module.Install() |
    | Remove     | $Module.Remove()  |
    ----------------------------------

    -----------------------------------------------------------------------------------------------------
    | Todo                                                                                              |
    |---------------------------------------------------------------------------------------------------|
    | PS Core         | Filter out stuff for PS Core, by building a different manifest                  |
    | PS Server       | Filter out stuff for PS Server, **                                              |
    -----------------------------------------------------------------------------------------------------

                                                                                                                 _______/
\_______________________________________________________________________________________________________________/ About
  Function /------------------------------------------------------------------------------------------------------------\
/----------
#>

Function FightingEntropy.Module
{
    [CmdLetBinding()]Param([Parameter()][UInt32]$Mode=0)

    # // =======================================================
    # // | Used to track console logging, similar to Stopwatch |
    # // =======================================================

    Class ConsoleTime
    {
        [String]   $Name
        [DateTime] $Time
        [UInt32]    $Set
        ConsoleTime([String]$Name)
        {
            $This.Name = $Name
            $This.Time = [DateTime]::MinValue
            $This.Set  = 0
        }
        Toggle()
        {
            $This.Time = [DateTime]::Now
            $This.Set  = 1
        }
        [String] ToString()
        {
            Return $This.Time.ToString()
        }
    }

    # // ========================================
    # // | Single object that displays a status |
    # // ========================================

    Class ConsoleItem
    {
        [UInt32]   $Index
        [String] $Elapsed
        [Int32]    $State
        [String]  $Status
        ConsoleItem([UInt32]$Index,[String]$Time,[Int32]$State,[String]$Status)
        {
            $This.Index   = $Index
            $This.Elapsed = $Time
            $This.State   = $State
            $This.Status  = $Status
        }
        [String] ToString()
        {
            Return "[{0}] (State: {1}/Status: {2})" -f $This.Elapsed, $This.State, $This.Status
        }
    }

    # // ========================================================================
    # // | A collection of status objects, uses itself to create/update messages |
    # // ========================================================================

    Class ConsoleController
    {
        [Object]    $Start
        [Object]      $End
        [String]     $Span
        [Object]   $Status
        [Object]   $Output
        ConsoleController()
        {
            $This.Reset()
        }
        [String] Elapsed()
        {
            $Item = Switch ($This.End.Set)
            {
                0 { [Timespan]([DateTime]::Now-$This.Start.Time) }
                1 { [Timespan]($This.End.Time-$This.Start.Time) }
            }

            Return $Item
        }
        [Object] ConsoleItem([Int32]$State,[String]$Status)
        {
            Return [ConsoleItem]::New($This.Output.Count,$This.Elapsed(),$State,$Status)
        }
        [Object] ConsoleTime([String]$Type)
        {
            Return [ConsoleTime]::New($Type)
        }
        Initialize()
        {
            Switch ($This.Start.Set)
            {
                0
                {
                    $This.Start.Toggle()
                    $This.Update(0,"Running [~] ($($This.Start))")
                }
                1
                {
                    $This.Update(-1,"Start [!] Error: Already initialized, try a different operation or reset.")
                }
            }
        }
        Finalize()
        {
            Switch ($This.Start.Set)
            {
                0
                {
                    $This.End.Toggle()
                    $This.Span = $This.Elapsed()
                    $This.Update(100,"Complete [+] ($($This.End)), Total: ($($This.Span))")
                }
                1
                {
                    $This.Update(-1,"End [!] Error: Already initialized, try a different operation or reset.")
                }
            }
        }
        Reset()
        {
            $This.Start  = $This.ConsoleTime("Start")
            $This.End    = $This.ConsoleTime("End")
            $This.Span   = $Null
            $This.Status = $Null
            $This.Output = [System.Collections.ObjectModel.ObservableCollection[Object]]::New()
        }
        Write()
        {
            $This.Output.Add($This.Status)
        }
        [Void] SetStatus([Int32]$State,[String]$Status)
        {
            $This.Status = $This.ConsoleItem($State,$Status)
        }
        [Object] Update([Int32]$State,[String]$Status)
        {
            $This.SetStatus($State,$Status)
            $This.Write()
            Return $This.Last()
        }
        [Object] Current()
        {
            $This.Update($This.Status.State,$This.Status.Status)
            Return $This.Last()
        }
        [Object] Last()
        {
            Return $This.Output[$This.Output.Count-1]
        }
        [Object] DumpConsole()
        {
            Return $This.Output | % ToString
        }
        [String] ToString()
        {
            Return @($This.Elapsed(),$This.Span)[!!$This.Span]
        }
    }

    # // ____________________________________________________________________
    # // | This is a 1x[track] x 4[char] chunk of information for Write-Host |
    # // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

    Class ThemeBlock
    {
        [UInt32]   $Index
        [Object]  $String
        [UInt32]    $Fore
        [UInt32]    $Back
        [UInt32]    $Last
        ThemeBlock([Int32]$Index,[String]$String,[Int32]$Fore,[Int32]$Back)
        {
            $This.Index  = $Index
            $This.String = $String
            $This.Fore   = $Fore
            $This.Back   = $Back
            $This.Last   = 1
        }
        Write([UInt32]$0,[UInt32]$1,[UInt32]$2,[UInt32]$3)
        {
            $Splat = @{

                Object          = $This.String
                ForegroundColor = @($0,$1,$2,$3)[$This.Fore]
                BackgroundColor = $This.Back
                NoNewLine       = $This.Last
            }

            Write-Host @Splat
        }
        [String] ToString()
        {
            Return "<FightingEntropy.Module.ThemeBlock>"
        }
    }

    # // ______________________________________________
    # // | Represents a 1x[track] in a stack of tracks |
    # // ----------------------------------------------

    Class ThemeTrack
    {
        [UInt32] $Index
        [Object] $Content
        ThemeTrack([UInt32]$Index,[Object]$Track)
        {
            $This.Index   = $Index
            $This.Content = $Track
        }
        [String] ToString()
        {
            Return "<FightingEntropy.Module.ThemeTrack>"
        }
    }

    # // ____________________________________________
    # // | Generates an actionable write-host object |
    # // --------------------------------------------

    Class ThemeStack
    {
        Hidden [Object]  $Face
        Hidden [Object] $Track
        ThemeStack([UInt32]$Slot,[String]$Message)
        {
            $This.Main($Message)
            $Object = $This.Palette($Slot)
            $This.Write($Object)
        }
        ThemeStack([String]$Message)
        {
            $This.Main($Message)
            $Object = $This.Palette(0)
            $This.Write($Object)
        }
        Main([String]$Message)
        {
            $This.Face = $This.Mask()
            $This.Reset()
            $This.Insert($Message)
        }
        [UInt32[]] Palette([UInt32]$Slot)
        {
            If ($Slot -gt 35)
            {
                Throw "Invalid entry"
            }

            Return @( Switch ($Slot)
            {
                00 {10,12,15,00} 01 {12,04,15,00} 02 {10,02,15,00} # Default, R*/Error,   G*/Success
                03 {01,09,15,00} 04 {03,11,15,00} 05 {13,05,15,00} # B*/Info, C*/Verbose, M*/Feminine
                06 {14,06,15,00} 07 {00,08,15,00} 08 {07,15,15,00} # Y*/Warn, K*/Evil,    W*/Host
                09 {04,12,15,00} 10 {12,12,15,00} 11 {04,04,15,00} # R!,      R+,         R-
                12 {02,10,15,00} 13 {10,10,15,00} 14 {02,02,15,00} # G!,      G+,         G-
                15 {09,01,15,00} 16 {09,09,15,00} 17 {01,01,15,00} # B!,      B+,         B-
                18 {11,03,15,00} 19 {11,11,15,00} 20 {03,03,15,00} # C!,      C+,         C-
                21 {05,13,15,00} 22 {13,13,15,00} 23 {05,05,15,00} # M!,      M+,         M-
            24 {06,14,15,00} 25 {14,14,15,00} 26 {06,06,15,00} # Y!,      Y+,         Y-
            27 {08,00,15,00} 28 {08,08,15,00} 29 {00,00,15,00} # K!,      K+,         K-
            30 {15,07,15,00} 31 {15,15,15,00} 32 {07,07,15,00} # W!,      W+,         W-
            33 {11,06,15,00} 34 {06,11,15,00} 35 {11,12,15,00} # Steel*,  Steel!,     C+R+
        })
    }
    [Object] Mask()
    {
        Return ("20202020 5F5F5F5F AFAFAFAF 2020202F 5C202020 2020205C 2F202020 5C5F5F2F "+
        "2FAFAF5C 2FAFAFAF AFAFAF5C 5C5F5F5F 5F5F5F2F 205F5F5F" -Split " ") | % { $This.Convert($_) }
    }
    [String] Convert([String]$Line)
    {
        Return [Char[]]@(0,2,4,6 | % { "0x$($Line.Substring($_,2))" | IEX }) -join ''
    }
    Add([String]$Mask,[String]$Fore)
    {
        # // ____________________________
        # // | Expands the mask strings |
        # // ----------------------------

        $Object        = Invoke-Expression $Mask | % { $This.Face[$_] }
        $FG            = Invoke-Expression $Fore
        $BG            = @(0)*30

        # // ____________________________
        # // | Generates a track object |
        # // ----------------------------

        $Hash          = @{ }
        ForEach ($X in 0..($Object.Count-1))
        {
            $Item      = [ThemeBlock]::New($X,$Object[$X],$FG[$X],$BG[$X])
            If ($X -eq $Object.Count-1)
            {
                $Item.Last = 0
            }
            $Hash.Add($Hash.Count,$Item)
        }
        $This.Track  += [ThemeTrack]::New($This.Track.Count,$Hash[0..($Hash.Count-1)])
    }
    [Void] Reset()
    {
        $This.Track = @( )

        # // ____________________________
        # // | Generates default tracks |
        # // ----------------------------

        $This.Add("0,1,0+@(1)*25+0,0","@(0)*30")
        $This.Add("3,8,7,9+@(2)*23+10,11,0","0,1,0+@(1)*25+0,0")
        $This.Add("5,7,9,13+@(0)*23+12,8,4","0,1,1+@(2)*24+1,1,0")
        $This.Add("0,10,11+@(1)*23+12+8,7,6","0,0+@(1)*25+0,1,0")
        $This.Add("0,0+@(2)*25+0,2,0","@(0)*30")
    }
    Insert([String]$String)
    {
        $This.Reset()
        $String = " $String"
        Switch ($String.Length)
        {
            {$_ -lt 84}
            {
                $String += (@(" ") * (84 - ($String.Length+1)) -join '' )
            }
            {$_ -ge 84}
            {
                $String  = $String.Substring(0,84) + "..."
            }
        }
        $Array = [Char[]]$String
        $Hash  = @{ }
        $Block = ""
        ForEach ($X in 0..($Array.Count-1))
        {
            If ($X % 4 -eq 0 -and $Block -ne "")
            {
                $Hash.Add($Hash.Count,$Block)
                $Block = ""
            }
            $Block += $Array[$X]
        }

        ForEach ($X in 0..($Hash.Count-1))
        {
            $This.Track[2].Content[$X+3].String = $Hash[$X]
        }
    }
    [Void] Write([UInt32[]]$Palette)
    {
        $0,$1,$2,$3 = $Palette
        ForEach ($Track in $This.Track)
        {
            ForEach ($Item in $Track.Content)
            {
               $Item.Write($0,$1,$2,$3)
            }
        }
    }
    [String] ToString()
    {
        Return "<FightingEntropy.Module.ThemeStack>"
    }
}

# // ____________________________________________________
# // | Property object which includes source and index  |
# // ----------------------------------------------------

Class OSProperty
{
    [String] $Source
    Hidden [UInt32] $Index
    [String] $Name
    [Object] $Value
    OSProperty([String]$Source,[UInt32]$Index,[String]$Name,[Object]$Value)
    {
        $This.Source = $Source
        $This.Index  = $Index
        $This.Name   = $Name
        $This.Value  = $Value
    }
    [String] ToString()
    {
        Return "<FightingEntropy.Module.OSProperty>"
    }
}

# // __________________________________________________________
# // | Container object for indexed OS (property/value) pairs |
# // ----------------------------------------------------------

Class OSPropertySet
{
    Hidden [UInt32] $Index
    [String] $Source
    [Object] $Property
    OSPropertySet([UInt32]$Index,[String]$Source)
    {
        $This.Index     = $Index
        $This.Source    = $Source
        $This.Property  = @( )
    }
    Add([String]$Name,[Object]$Value)
    {
        $This.Property += [OSProperty]::New($This.Source,$This.Property.Count,$Name,$Value)
    }
    [String] ToString()
    {
        $D = ([String]$This.Property.Count).Length
        Return "({0:d$D}) <FightingEntropy.Module.OSPropertySet[{1}]>" -f $This.Property.Count, $This.Source
    }
}


# // _______________________________________________________
# // | Collects various details about the operating system |
# // | specifically for cross-platform compatibility       |
# // ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾

Class OS
{
    Hidden [String] $Name
    [Object]     $Caption
    [Object]    $Platform
    [Object]   $PSVersion
    [Object]        $Type
    [Object]      $Output
    OS()
    {
        $This.Name   = "Operating System"
        $This.Output = @( )

        # // _______________
        # // | Environment |
        # // ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾

        $This.AddPropertySet("Environment")

        Get-ChildItem Env:              | % { $This.Add(0,$_.Key,$_.Value) }

        # // ____________
        # // | Variable |
        # // ‾‾‾‾‾‾‾‾‾‾‾‾

        $This.AddPropertySet("Variable")

        Get-ChildItem Variable:         | % { $This.Add(1,$_.Name,$_.Value) }

        # // ________
        # // | Host |
        # // ‾‾‾‾‾‾‾‾

        $This.AddPropertySet("Host")

        (Get-Host).PSObject.Properties  | % { $This.Add(2,$_.Name,$_.Value) }

        # // ______________
        # // | PowerShell |
        # // ‾‾‾‾‾‾‾‾‾‾‾‾‾‾

        $This.AddPropertySet("PowerShell")

        (Get-Variable PSVersionTable | % Value).GetEnumerator() | % { $This.Add(3,$_.Name,$_.Value) }

        If ($This.Tx("PowerShell","PSedition") -eq "Desktop")
        {
            Get-CimInstance Win32_OperatingSystem | % { $This.Add(3,"OS","Microsoft Windows $($_.Version)") }
            $This.Add(3,"Platform","Win32NT")
        }

        # // ____________________________________
        # // | Assign hashtable to output array |
        # // ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾

        $This.Caption   = $This.Tx("PowerShell","OS")
        $This.Platform  = $This.Tx("PowerShell","Platform")
        $This.PSVersion = $This.Tx("PowerShell","PSVersion")
        $This.Type      = $This.GetOSType()
    }
    [Object] Tx([String]$Source,[String]$Name)
    {
        Return $This.Output | ? Source -eq $Source | % Property | ? Name -eq $Name | % Value
    }
    Add([UInt32]$Index,[String]$Name,[Object]$Value)
    {
        $This.Output[$Index].Add($Name,$Value)
    }
    AddPropertySet([String]$Name)
    {
        $This.Output += [OSPropertySet]::New($This.Output.Count,$Name)
    }
    [String] GetWinCaption()
    {
        Return "[wmiclass]'Win32_OperatingSystem' | % GetInstances | % Caption"
    }
    [String] GetWinType()
    {
        Return @(Switch -Regex (Invoke-Expression $This.GetWinCaption())
        {
            "Windows (10|11)" { "Win32_Client" } "Windows Server" { "Win32_Server" }
        })
    }
    [String] GetOSType()
    {
        Return @( If ($This.Version.Major -gt 5)
        {
            If (Get-Item Variable:\IsLinux | % Value)
            {
                (hostnamectl | ? { $_ -match "Operating System" }).Split(":")[1].TrimStart(" ")
            }

            Else
            {
                $This.GetWinType()
            }
        }

        Else
        {
            $This.GetWinType()
        })
    }
    [String] ToString()
    {
        Return "<FightingEntropy.Module.OS>"
    }
}

# // ____________________________________________________________
# // | Meant to determine longest file name and provide spacing |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

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

# // ______________________________________________________________
# // | Manifest file -> filesystem object (collection/validation) |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class ManifestFile
{
    Hidden [UInt32]    $Index
    Hidden [UInt32]     $Mode
    [String]            $Type
```

```
[String]            $Name
[String]            $Hash
[UInt32]          $Exists
Hidden [String] $Fullname
Hidden [String]   $Source
Hidden [UInt32]    $Match
Hidden [Object]  $Content
ManifestFile([Object]$Folder,[String]$Name,[String]$Hash,[String]$Source)
{
    $This.Index    = $Folder.Item.Count
    $This.Mode     = 0
    $This.Type     = $Folder.Type
    $This.Name     = $Name
    $This.Fullname = "{0}\$Name" -f $Folder.Fullname
    $This.Source   = "{0}/{1}/{2}?raw=true" -f $Source, $Folder.Name, $Name
    $This.Hash     = $Hash
    $This.TestPath()
}
TestPath()
{
    $This.Exists   = [System.IO.File]::Exists($This.Fullname)
}
[Void] Create()
{
    $This.TestPath()

    If (!$This.Exists)
    {
        [System.IO.File]::Create($This.Fullname).Dispose()
        $This.Exists = 1
    }
}
[Void] Remove()
{
    $This.TestPath()

    If ($This.Exists)
    {
        [System.IO.File]::Delete($This.Fullname)
        $This.Exists = 0
    }
}
Download()
{
    Try
    {
        $xContent = Invoke-WebRequest $This.Source -UseBasicParsing | % Content

        If ($This.Name -match "\.+(jpg|jpeg|png|bmp|ico)")
        {
            $This.Content = $xContent
        }
        ElseIf ($This.Name -match "\.+(txt|xml|cs)")
        {
            $Array = $xContent -Split "`n"
            $Ct    = $Array.Count
            Do
            {
                If ($Array[$Ct] -notmatch "\w")
                {
                    $Ct --
                }
            }
            Until ($Array[$Ct] -match "\w")

            $This.Content = $Array[0..($Ct)] -join "`n"
        }
        Else
        {
            $This.Content = $xContent
        }
    }
    Catch
    {
        Throw "Exception [!] An unspecified error occurred"
    }
}
```

```
    Write()
    {
        If (!$This.Content)
        {
            Throw "Exception [!] Content not assigned, cannot (write/set) content."
        }

        If (!$This.Exists)
        {
            Throw "Exception [!] File does not exist."
        }

        Try
        {
            If ($This.Name -match "\.+(jpg|jpeg|png|bmp|ico)")
            {
                [System.IO.File]::WriteAllBytes($This.Fullname,[Byte[]]$This.Content)
            }
            ElseIf ($This.Name -match "\.+(txt|xml|cs)")
            {
                [System.IO.File]::WriteAllText($This.Fullname,$This.Content)
            }
            Else
            {
                [System.IO.File]::WriteAllText($This.Fullname,
                                               $This.Content,
                                               [System.Text.UTF8Encoding]$False)
            }
        }
        Catch
        {
            Throw "Exception [!] An unspecified error has occurred"
        }
    }
    GetContent()
    {
        If (!$This.Exists)
        {
            Throw "Exception [!] File does not exist, it needs to be created first."
        }

        Try
        {
            If ($This.Name -match "\.+(jpg|jpeg|png|bmp|ico)")
            {
                $This.Content = [System.IO.File]::ReadAllBytes($This.Fullname)
            }
            ElseIf ($This.Name -match "\.+(xml|txt|cs)")
            {
                $This.Content = [System.IO.File]::ReadAllText($This.Fullname,
                                                              [System.Text.UTF8Encoding]$False)
            }
            Else
            {
                $This.Content = [System.IO.File]::ReadAllLines($This.Fullname,
                                                               [System.Text.UTF8Encoding]$False)
            }
        }
        Catch
        {
            Throw "Exception [!] An unspecified error has occurred"
        }
    }
    [String] ToString()
    {
        Return "<FightingEntropy.Module.ManifestFile>"
    }
}

# // ________________________________________
# // | Manifest folder -> filesystem object |
# // ----------------------------------------

Class ManifestFolder
{
    Hidden [UInt32]    $Index
    Hidden [UInt32]     $Mode
```

```powershell
    [String]            $Type
    [String]            $Name
    [String]        $Fullname
    [UInt32]          $Exists
    Hidden [Object]     $Item
    Hidden [String]   $Source
    ManifestFolder([UInt32]$Index,[String]$Type,[String]$Parent,[String]$Name)
    {
        $This.Index     = $Index
        $This.Mode      = 1
        $This.Type      = $Type
        $This.Name      = $Name
        $This.Fullname  = "$Parent\$Name"
        $This.Item      = @( )
        $This.TestPath()
    }
    Add([Object]$File)
    {
        If ($File.Exists)
        {
            $Hash       = Get-FileHash $File.Fullname | % Hash
            If ($Hash -eq $File.Hash)
            {
                $File.Match = 1
            }
            If ($Hash -ne $File.Hash)
            {
                $File.Match = 0
            }
        }

        $This.Item     += $File
    }
    [Object] Get([String]$Name)
    {
        Return $This.Output | ? Name -eq $Name
    }
    TestPath()
    {
        If (!$This.Fullname)
        {
            Throw "Exception [!] Resource path not set"
        }

        $This.Exists = [System.IO.Directory]::Exists($This.Fullname)
    }
    [Void] Create()
    {
        $This.TestPath()

        If (!$This.Exists)
        {
            [System.IO.Directory]::CreateDirectory($This.Fullname)
            $This.Exists = 1
        }
    }
    [Void] Remove()
    {
        $This.TestPath()

        If ($This.Exists)
        {
            [System.IO.Directory]::Delete($This.Fullname)
            $This.Exists = 0
        }
    }
    [String] ToString()
    {
        Return "({0}) <FightingEntropy.Module.ManifestFolder[{1}]>" -f $This.Item.Count, $This.Name
    }
}

# // _____________________________________________________________________
# // | File manifest container, laid out for hash (insertion+validation) |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class ManifestController
```

```
{
    Hidden [String]    $Name
    [String]         $Source
    [String]       $Resource
    Hidden [UInt32]   $Depth
    Hidden [UInt32]   $Total
    [Object]         $Output
    ManifestController([String]$Source,[String]$Resource)
    {
        $This.Name     = "Module Manifest"
        $This.Source   = $Source
        $This.Resource = $Resource
        $This.Output   = @( )
    }
    [Object] Get([String]$Name)
    {
        Return $This.Output | ? Name -eq $Name | % Output
    }
    [Object[]] Refresh()
    {
        $Out = @( )
        ForEach ($List in $This.Output)
        {
            $List.TestPath()
            $Out += $List
            If ($List.Exists)
            {
                ForEach ($Item in $List.Item)
                {
                    $Item.TestPath()
                    $Out += $Item
                }
            }
        }

        Return $Out
    }
    [Object] Files([UInt32]$Index)
    {
        Return $This.Output[$Index] | % Item
    }
    [Object] Full()
    {
        $D = "Index Type Name Hash Exists Fullname Source Match" -Split " "
        Return $This.Output | % Item | Select-Object $D
    }
    [String] ToString()
    {
        Return "<FightingEntropy.Module.ManifestController>"
    }
}

# // ___________________________________
# // | Template for registry injection |
# // -----------------------------------

Class RegistryTemplate
{
    [String]      $Source
    [String]        $Name
    [String] $Description
    [String]      $Author
    [String]     $Company
    [String]   $Copyright
    [Guid]          $Guid
    [DateTime]      $Date
    [String]     $Version
    [String]     $Caption
    [String]    $Platform
    [String]        $Type
    [String]    $Registry
    [String]    $Resource
    [String]      $Module
    [String]        $File
    [String]    $Manifest
    RegistryTemplate([Object]$Module)
    {
        $This.Source      = $Module.Source
        $This.Name        = $Module.Name
        $This.Description = $Module.Description
        $This.Author      = $Module.Author
        $This.Company     = $Module.Company
        $This.Copyright   = $Module.Copyright
        $This.Guid        = $Module.Guid
        $This.Date        = $Module.Date
        $This.Version     = $Module.Version
        $This.Caption     = $Module.OS.Caption
        $This.Platform    = $Module.OS.Platform
        $This.Type        = $Module.OS.Type
        $This.Registry    = $Module.Root.Registry
        $This.Resource    = $Module.Root.Resource
        $This.Module      = $Module.Root.Module
        $This.File        = $Module.Root.File
        $This.Manifest    = $Module.Root.Manifest
    }
}

# // __________________________________________________
# // | Represents individual paths to the module root |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class RootProperty
{
    Hidden [UInt32] $Index
    [String]         $Type
    [String]         $Name
    [String]     $Fullname
    [UInt32]       $Exists
    Hidden [String]  $Path
    RootProperty([UInt32]$Index,[String]$Name,[UInt32]$Type,[String]$Fullname)
    {
        $This.Index    = $Index
        $This.Type     = Switch ($Type) { 0 { "Directory" } 1 { "File" } }
        $This.Name     = $Name
        $This.Fullname = $Fullname
        $This.Path     = $Fullname
        $This.TestPath()
    }
    TestPath()
    {
        $This.Exists   = Test-Path $This.Path
    }
    Create()
    {
        $This.TestPath()

        If (!$This.Exists)
        {
            Switch ($This.Name)
            {
                {$_ -in "Resource","Module"}
                {
                    [System.IO.Directory]::CreateDirectory($This.Fullname)
                }
                {$_ -in "File","Manifest"}
                {
                    [System.IO.File]::Create($This.Fullname).Dispose()
                }
            }

            $This.TestPath()
        }
    }
    Remove()
    {
        $This.TestPath()

        If ($This.Exists)
        {
            Switch ($This.Name)
            {
                {$_ -in "Resource","Module"}
                {
                    [System.IO.Directory]::Delete($This.Fullname)
                }
                {$_ -in "File","Manifest","Shortcut"}
                {
                    [System.IO.File]::Delete($This.Fullname)
                }
            }

            $This.Exists = 0
        }
    }
    [String] ToString()
    {
        Return $This.Path
    }
}

# // ________________________________________________________
# // | Represents a collection of paths for the module root |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class Root
{
    Hidden [String] $Name
    [Object]    $Registry
    [Object]    $Resource
    [Object]      $Module
    [Object]        $File
    [Object]    $Manifest
    [Object]    $Shortcut
    Root([String]$Version,[String]$Resource,[String]$Path)
    {
        $This.Name     = "Module Root"
        $SDP           = "Secure Digits Plus LLC"
        $FE            = "FightingEntropy"
        $This.Registry = $This.Set(0,0,"HKLM:\Software\Policies\$SDP\$FE\$Version")
        $This.Resource = $This.Set(1,0,"$Resource")
        $This.Module   = $This.Set(2,0,"$Path\$FE")
        $This.File     = $This.Set(3,1,"$Path\$FE\$FE.psm1")
        $This.Manifest = $This.Set(4,1,"$Path\$FE\$FE.psd1")
        $This.Shortcut = $This.Set(5,1,"$Env:Public\Desktop\$FE.lnk")
    }
    [String] Slot([UInt32]$Type)
    {
        Return @("Registry","Resource","Module","File","Manifest","Shortcut")[$Type]
    }
    [Object] Set([UInt32]$Index,[UInt32]$Type,[String]$Path)
    {
        Return [RootProperty]::New($Index,$This.Slot($Index),$Type,$Path)
    }
    [Void] Refresh()
    {
        $This.List() | % { $_.TestPath() }
    }
    [Object[]] List()
    {
        Return $This.PSObject.Properties.Name | % { $This.$_ }
    }
    [String] ToString()
    {
        Return "<FightingEntropy.Module.Root>"
    }
}

# // ___________________________________________
# // | Works as a PowerShell Registry provider |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class RegistryKeyTemp
{
    Hidden [Microsoft.Win32.RegistryKey] $Key
    Hidden [Microsoft.Win32.RegistryKey] $Subkey
    [String]            $Enum
    [String]            $Hive
    [String]            $Path
    [String]            $Name
    Hidden [String] $Fullname
    RegistryKeyTemp([String]$Path)
```

```
        {
            $This.Fullname = $Path
            $Split         = $Path -Split "\\"
            $This.Hive     = $Split[0]
            $This.Name     = $Split[-1]
            $This.Enum     = Switch -Regex ($This.Hive)
            {
                HKLM: {"LocalMachine"} HKCU: {"CurrentUser"} HKCR: {"ClassesRoot"}
            }
            $This.Path     = $Path -Replace "$($This.Hive)\\", "" | Split-Path -Parent
        }
        Open()
        {
            $X             = $This.Enum
            $This.Key      = [Microsoft.Win32.Registry]::$X.CreateSubKey($This.Path)
        }
        Create()
        {
            If (!$This.Key)
            {
                Throw "Must open the key first."
            }

            $This.Subkey = $This.Key.CreateSubKey($This.Name)
        }
        Add([String]$Name,[Object]$Value)
        {
            If (!$This.Subkey)
            {
                Throw "Must create the subkey first."
            }

            $This.Subkey.SetValue($Name,$Value)
        }
        [Void] Remove()
        {
            If ($This.Key)
            {
                $This.Key.DeleteSubKeyTree($This.Name)
            }
        }
        [Void] Dispose()
        {
            If ($This.Subkey)
            {
                $This.Subkey.Flush()
                $This.Subkey.Dispose()
            }

            If ($This.Key)
            {
                $This.Key.Flush()
                $This.Key.Dispose()
            }
        }
    }

    # // ________________________________________________________
    # // | Represents an individual registry key for the module |
    # // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

    Class RegistryKeyProperty
    {
        Hidden [UInt32] $Index
        [String]         $Name
        [Object]        $Value
        [UInt32]       $Exists
        RegistryKeyProperty([UInt32]$Index,[Object]$Property)
        {
            $This.Index = $Index
            $This.Name  = $Property.Name
            $This.Value = $Property.Value
        }
        [String] ToString()
        {
            Return "<FightingEntropy.Module.RegistryKeyProperty>"
        }
```

```
}

# // _________________________________________________________
# // | Represents a collection of registry keys for the module |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class RegistryKey
{
    Hidden [String] $Name
    [String]        $Path
    [UInt32]      $Exists
    [Object]    $Property
    RegistryKey([Object]$Module)
    {
        $This.Name         = "Module Registry"
        $This.Path         = $Module.Root.Registry.Path
        $This.TestPath()
        If ($This.Exists)
        {
            $Object        = Get-ItemProperty $This.Path
            $This.Property = $This.Inject($Object)
        }
        Else
        {
            $Object        = $Module.Template()
            $This.Property = $This.Inject($Object)
        }
    }
    [Object] Inject([Object]$Object)
    {
        $Hash              = @{ }
        ForEach ($Property in $Object.PSObject.Properties | ? Name -notmatch ^PS)
        {
            $Item          = $This.Key($Hash.Count,$Property)
            $Item.Exists   = $This.Exists
            $Hash.Add($Hash.Count,$Item)
        }

        Return $Hash[0..($Hash.Count-1)]
    }
    TestPath()
    {
        $This.Exists = Test-Path $This.Path
    }
    Create()
    {
        $This.TestPath()

        If ($This.Exists)
        {
            Throw "Exception [!] Path already exists"
        }

        $Key            = $This.RegistryKeyTemp($This.Path)
        $Key.Open()
        $Key.Create()

        $This.Exists    = 1

        ForEach ($X in 0..($This.Property.Count-1))
        {
            $Item        = $This.Property[$X]
            $Key.Add($Item.Name,$Item.Value)
            $Item.Exists = 1
        }
        $Key.Dispose()
    }
    Remove()
    {
        $This.TestPath()

        If (!$This.Exists)
        {
            Throw "Exception [!] Registry path does not exist"
        }

        $Key             = $This.RegistryKeyTemp($This.Path)
        $Key.Open()
        $Key.Create()
        $Key.Delete()

        ForEach ($Item in $This.Property)
        {
            $Item.Exists = 0
        }

        $This.Exists     =   0
        $Key.Dispose()
    }
    [Object[]] List()
    {
        Return $This.Output
    }
    [Object] Key([UInt32]$Index,[Object]$Property)
    {
        Return [RegistryKeyProperty]::New($Index,$Property)
    }
    [Object] KeyTemp([String]$Path)
    {
        Return [RegistryKeyTemp]::New($Path)
    }
    [String] ToString()
    {
        Return "<FightingEntropy.Module.RegistryKey>"
    }
}

# // ___________________________________________
# // | Collects/creates versions of the module |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class FEVersion
{
    [Version]      $Version
    Hidden [DateTime] $Time
    [String]          $Date
    [Guid]            $Guid
    FEVersion([String]$Line)
    {
        $This.Version = $This.Tx(0,$Line)
        $This.Time    = $This.Tx(1,$Line)
        $This.Date    = $This.MilitaryTime()
        $This.Guid    = $This.Tx(2,$Line)
    }
    FEVersion([Switch]$New,[String]$Version)
    {
        $This.Version = $Version
        $This.Time    = [DateTime]::Now
        $This.Date    = $This.MilitaryTime()
        $This.Guid    = [Guid]::NewGuid()
    }
    [String] MilitaryTime()
    {
        Return $This.Time.ToString("MM/dd/yyyy HH:mm:ss")
    }
    [String] Tx([UInt32]$Type,[String]$Line)
    {
        $Pattern = Switch ($Type)
        {
            0 { "\d{4}\.\d{2}\.\d+" }
            1 { "\d{2}\/\d{2}\/\d{4} \d{2}:\d{2}:\d{2}" }
            2 { @(8,4,4,4,12 | % { "[a-f0-9]{$_}" }) -join '-' }
        }

        Return [Regex]::Matches($Line,$Pattern).Value
    }
    [String] ToString()
    {
        Return "| {0} | {1} | {2} |" -f $This.Version,
                                        $This.Date.ToString("MM/dd/yyyy HH:mm:ss"),
                                        $This.Guid
    }
}
```

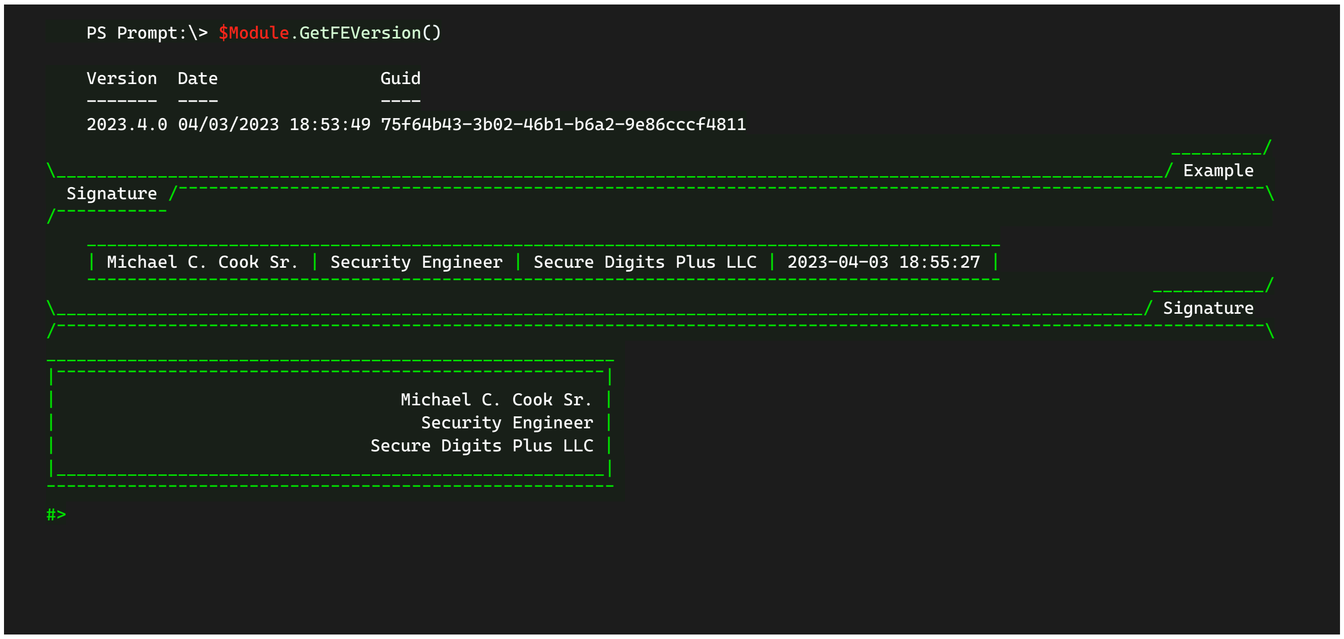
```
# // ________________________________________________________
# // | Specifically used for file hash validation/integrity |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class ValidateFile
{
    [UInt32]           $Index
    [String]            $Type
    [String]            $Name
    [String]            $Hash
    [String]         $Current
    Hidden [String] $Fullname
    Hidden [String]   $Source
    [UInt32]          $Exists
    [UInt32]           $Match
    ValidateFile([Object]$File)
    {
        $This.Index    = $File.Index
        $This.Type     = $File.Type
        $This.Name     = $File.Name
        $This.Hash     = $File.Hash
        $This.Current  = $This.GetFileHash($File.Fullname)
        $This.Exists   = $File.Exists
        $This.Fullname = $File.Fullname
        $This.Source   = $File.Source
        $This.Match    = [UInt32]($This.Hash -eq $This.Current)
        $File.Match    = $This.Match
    }
    [String] GetFileHash([String]$Path)
    {
        If (![System.IO.File]::Exists($Path))
        {
            Throw "Invalid path"
        }

        Return Get-FileHash $Path | % Hash
    }
}

# // ______________________________________________________________
# // | Factory class to control all of the aforementioned classes |
# // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Class InstallController
{
    Hidden [UInt32]    $Mode
    Hidden [Object] $Console
    [String]         $Source = "https://www.github.com/mcc85s/FightingEntropy"
    [String]           $Name = "[FightingEntropy($([Char]960))]"
    [String]    $Description = "Beginning the fight against ID theft and cybercrime"
    [String]         $Author = "Michael C. Cook Sr."
    [String]        $Company = "Secure Digits Plus LLC"
    [String]      $Copyright = "(c) 2023 (mcc85s/mcc85sx/sdp). All rights reserved."
    [Guid]             $Guid = "75f64b43-3b02-46b1-b6a2-9e86cccf4811"
    [DateTime]         $Date = "04/03/2023 18:53:49"
    [Version]       $Version = "2023.4.0"
    [Object]             $OS
    [Object]           $Root
    [Object]       $Manifest
    [Object]       $Registry
    InstallController([Switch]$Flags)
    {
        $This.Mode = 0
        $This.Main()
    }
    InstallController()
    {
        $This.Mode = 0
        $This.Main()
    }
    InstallController([UInt32]$Mode)
    {
        $This.Mode = $Mode
        $This.Main()
    }
    Main()
    {
        # Initialize console
        $This.StartConsole()

        # Display module
        $This.Display()

        # Operating system
        $This.OS       = $This.New("OS")

        # Root
        $This.Root     = $This.New("Root")

        # Manifest
        $This.Manifest = $This.New("Manifest")

        # Registry
        $This.Registry = $This.New("Registry")

        $This.Update(0,"
  ")

        # Load the manifest
        $This.LoadManifest()
    }
    StartConsole()
    {
        # Instantiates and initializes the console
        $This.Console = [ConsoleController]::New()
        $This.Console.Initialize()
        $This.Status()
    }
    [Void] Status()
    {
        # If enabled, shows the last item added to the console
        If ($This.Mode -eq 0)
        {
            [Console]::WriteLine($This.Console.Last().Status)
        }
    }
    [Void] Update([Int32]$State,[String]$Status)
    {
        # Updates the console
        $This.Console.Update($State,$Status)
        $This.Status()
    }
    [Void] Write([String]$Message)
    {
        # Writes a standard stylized message to the console
        [ThemeStack]::New($Message)
    }
    [Void] Write([UInt32]$Slot,[String]$Message)
    {
        # Writes a selected stylized message to the console
        [ThemeStack]::New($Slot,$Message)
    }
    Display()
    {
        If ($This.Mode -eq 0)
        {
            $This.Update(0,"Loading [~] $($This.Label())")
            $This.Write($This.Console.Last().Status)
        }
    }
    [String] Label()
    {
        # Returns the module name and version as a string
        Return "{0}[{1}]" -f $This.Name, $This.Version.ToString()
    }
    [String] SourceUrl()
    {
        # Returns the (base url + version) as a string
        Return "{0}/blob/main/Version/{1}" -f $This.Source, $This.Version
    }
    [String] Env([String]$Name)
    {
        # Returns named environment variable as a string
        Return [Environment]::GetEnvironmentVariable($Name)
```

```
}
[String] GetResource()
{
    # Returns the resource path as a string
    Return $This.Env("ProgramData"), $This.Company, "FightingEntropy", $This.Version.ToString() -join "\"
}
[String] GetRootPath()
{
    # Selects and returns the root module path as a string
    $Path     = Switch -Regex ($This.OS.Type)
    {
        ^Win32_ { $This.Env("PSModulePath") -Split ";" -match [Regex]::Escape($This.Env("Windir")) }
        Default { $This.Env("PSModulePath") -Split ":" -match "PowerShell"                         }
    }

    Return $Path
}
[Object] GetFEVersion()
{
    # Returns parsed FEModule version object
    Return [FEVersion]::New("| $($This.Version) | $($This.Date) | $($This.Guid) |")
}
[Object] ManifestFolder([UInt32]$Index,[String]$Type,[String]$Resource,[String]$Name)
{
    # Instantiates a new manifest folder, and can be used externally
    Return [ManifestFolder]::New($Index,$Type,$Resource,$Name)
}
[Object] ManifestFile([Object]$Folder,[String]$Name,[String]$Hash)
{
    # Instantiates a new manifest file, and can be used externally
    Return [ManifestFile]::New($Folder,$Name,$Hash,$This.SourceUrl())
}
[Object] NewVersion([String]$Version)
{
    # Tests a version input string, and if it passes, returns a version object
    If ($Version -notmatch "\d{4}\.\d{1,}\.\d{1,}")
    {
        Throw "Invalid version entry"
    }

    Return [FEVersion]::New($True,$Version)
}
[Object[]] Versions()
{
    # Obtains the available versions from the project site
    $Markdown = Invoke-RestMethod "$($This.Source)/blob/main/README.md?raw=true"
    Return $Markdown -Split "`n" | ? { $_ -match "^\|\s\*\*\d{4}\.\d{2}\.\d+\*\*" } | % { [FEVersion]$_ }
}
[Object] Template()
{
    # Instantiates a new registry template to generate a registry key set
    Return [RegistryTemplate]::New($This)
}
[Object] New([String]$Name)
{
    # (Selects/instantiates) selected object
    $Item = Switch ($Name)
    {
        OS
        {
            [OS]::New()
        }
        Root
        {
            [Root]::New($This.Version,$This.GetResource(),$This.GetRootPath())
        }
        Manifest
        {
            [ManifestController]::New($This.Source,$This.Root.Resource)
        }
        Registry
        {
            [RegistryKey]::New($This)
        }
    }

    # Logs the instantiation of the named (function/class)
```

```
            Switch ([UInt32]!!$Item)
            {
                0 { $This.Update(-1,"[!] <$($Item.Name)> ") }
                1 { $This.Update( 1,"[+] <$($Item.Name)> ") }
            }

            Return $Item
        }
        [Object] GetFolder([String]$Type)
        {
            # Returns the named folder from the manifest controller
            Return $This.Manifest.Output | ? Type -eq $Type
        }
        [Object] GetFolder([UInt32]$Index)
        {
            # Returns the indexed folder from the manifest controller
            Return $This.Manifest.Output | ? Index -eq $Index
        }
        [String] GetFolderName([String]$Type)
        {
            # Returns the formal name of a given (type/folder) as a string
            $xName = Switch ($Type)
            {
                Control  {   "Control" }
                Function { "Functions" }
                Graphic  {  "Graphics" }
            }

            Return $xName
        }
        [Object] ManifestListItem([UInt32]$Index,[String]$Source,[String]$Name,[String]$Hash)
        {
            Return [ManifestListItem]::New($Index,$Source,$Name,$Hash)
        }
        [Object[]] GetManifestList([String]$Name)
        {
            $List = Switch ($Name)
            {
                Control
                {
                    ("Computer.png"                    ,
"87EAB4F74B38494A960BEBF69E472AB0764C3C7E782A3F74111F993EA31D1075") ,
                    ("DefaultApps.xml"                 ,
"EEC0F0DFEAC1B4172880C9094E997C8A5C5507237EB70A241195D7F16B06B035") ,
                    ("failure.png"                     ,
"59D479A0277CFFDD57AD8B9733912EE1F3095404D65AB630F4638FA1F40D4E99") ,
                    ("FEClientMod.xml"                 ,
"326C8D3852895A3135144ACCBB4715D2AE49101DCE9E64CA6C44D62BD4F33D02") ,
                    ("FEServerMod.xml"                 ,
"3EA9AF3FFFB5812A3D3D42E5164A58EF2FC744509F2C799CE7ED6D0B0FF9016D") ,
                    ("header-image.png"                ,
"38F1E2D061218D31555F35C729197A32C9190999EF548BF98A2E2C2217BBCB88") ,
                    ("MDTClientMod.xml"                ,
"B2BA25AEB67866D17D8B22BFD31281AFFF0FFE1A7FE921A97C51E83BF46F8603") ,
                    ("MDTServerMod.xml"                ,
"C4B12E67357B54563AB042617CEC2B56128FD03A9C029D913BB2B6CC65802189") ,
                    ("MDT_LanguageUI.xml"              ,
"8968A07D56B4B2A56F15C07FC556432430CB1600B8B6BBB13C332495DEE95503") ,
                    ("PSDClientMod.xml"                ,
"C90146EECF2696539ACFDE5C2E08CFD97548E639ED7B1340A650C27F749AC9CE") ,
                    ("PSDServerMod.xml"                ,
"C90146EECF2696539ACFDE5C2E08CFD97548E639ED7B1340A650C27F749AC9CE") ,
                    ("success.png"                     ,
"46757AB0E2D3FFFFDBA93558A34AC8E36F972B6F33D00C4ADFB912AE1F6D6CE2") ,
                    ("vendorlist.txt"                  ,
"A37B6652014467A149AC6277D086B4EEE7580DDB548F81B0B2AA7AC78C240874") ,
                    ("Wifi.cs"                         ,
"40522623 4D7726180C0F9C97DF3C663CA0028A36CBCD00806D6517575A6F549F") ,
                    ("zipcode.txt"                     ,
"E471E887F537FA295A070AB41E21DEE978181A92CB204CA1080C6DC32CBBE0D8")
                }
                Function
                {
                    ("Copy-FileStream.ps1"             ,
"51D78BCE84D5EC2FABF85F87078D8E42179B19195E546371FC439E4B6171A0B9") ,
                    ("Get-AssemblyList.ps1"            ,
"F2D1C0AD58A91CBF432A2AC793C8CD1313EB6F1A61C50D681130322C358CDAE7") ,
                    ("Get-ControlExtension.ps1"        ,
"BF83DAAF1D8D53A39A5C5402A6BA9DEEA3DF32D37F38214DD93D1EBBE314942D") ,
                    ("Get-DCOMSecurity.ps1"            ,
"066C56A9EDC23D5D74E513C13A6C48F8B593D27189AB44B70FD53D6A9C3F965C") ,
                    ("Get-EnvironmentKey.ps1"          ,
"96F00FD11983FF80BCB62C70826DB9B1608C84448C68E9C52857A224CA0054F6") ,
                    ("Get-EventLogArchive.ps1"         ,
"D0FB5197A191B28BA5ABB1577A7C27A6684373C2FCC1F4E88628E2E4FDB72925") ,
                    ("Get-EventLogConfigExtension.ps1" ,
"F91B1681063A5142129E40DD7F77F2D99813B6089B4D45E6F0DE5AA28FA01099") ,
                    ("Get-EventLogController.ps1"      ,
"B270065C25EAB6183A10043858F56256059D070CC2E0D37A4352D379A36ACAF5") ,
                    ("Get-EventLogProject.ps1"         ,
"113E9EB104D983F1F990D738E1EA89E685B70270B6B856E16F16A40E8748CDE6") ,
                    ("Get-EventLogRecordExtension.ps1" ,
"8B738D1B551BC14C6FD8D003A82E420CDA17ED865FFD83D6E3A392F40CF20145") ,
                    ("Get-EventLogXaml.ps1"            ,
"18554029561A277AEB5AAA643CF88DC43F3A7C2D97281EEBD47A03BEE6018DB4") ,
                    ("Get-FEADLogin.ps1"               ,
"1EEA605D7181E9F1985FC012E7EABB1884B39B9D33D2E2E8AB6A8C21C3770B56") ,
                    ("Get-FEDCPromo.ps1"               ,
"0B682031192C18EC2F9135A664DADD254E45CDDAF36D863EB2E6760CB1379323") ,
                    ("Get-FEImageManifest.ps1"         ,
"03AD403FA17EE0702A8D8911F8B4BD7AABE5C6971363AF2FFADE6FF83918D57F") ,
                    ("Get-FEModule.ps1"                ,
"9E2CCD51F1082FC197ABCF68B8429FC62572DF3DA8AF5CAD29580F37C33C81DF") ,
                    ("Get-FENetwork.ps1"               ,
"0048A6208F9DDF0CCCFBCEE0621426DE2B49ACCBDBED71FB1E5D8B027330CEFC") ,
                    ("Get-FERole.ps1"                  ,
"0016BDDB9B0BA9BB59652440FE0B758D88BF42A887F93B275F57016CCE4999C8") ,
                    ("Get-FESystem.ps1"                ,
"A8A54664FCAEA3F59E387CAEF927F26009F20BC28C689417E6D840A062F166B0") ,
                    ("Get-MDTModule.ps1"               ,
"FC61D8D17B22A6AC2AE343A3EA7A07DBF868D918C85D302DF771862306CB824A") ,
                    ("Get-PowerShell.ps1"              ,
"7F5E35535A4A50D02092D8A87266F136EEBD979F9505D8D481A4F5E38E74BF02") ,
                    ("Get-PropertyItem.ps1"            ,
"48E4729380C40B76B13DE0FD6CAC735B05B76D78CE86636F9258D1F3D60AD6B0") ,
                    ("Get-PropertyObject.ps1"          ,
"7657A59EBD53E31AC69C3B48CAD83B52D6F22D8D4F12EBF4BE223DF315F5DAD2") ,
                    ("Get-PSDLog.ps1"                  ,
"2C7DC771C2BECE4DF20C41567E4944C836FC7D6592C3451DAA798010DC50CACB") ,
                    ("Get-PSDLogGUI.ps1"               ,
"FEBF687E9A97A413576DD515DE7184D4E71AA8EC61737A53EC39F5BDB11588FF") ,
                    ("Get-PSDModule.ps1"               ,
"CF59887548D790EE8B4D339450BFC1D64227F68CC4E555C877B9AFDD54CB5EBD") ,
                    ("Get-ThreadController.ps1"        ,
"66C2078C9CC0621CE911CCE301490BA36214CECC9415F982CC819651FD1E9E66") ,
                    ("Get-UserProfile.ps1"             ,
"F9A6B23DCE348E5627F62F1C7A53A1D7A73BB417AA8927E6A95CEE25142A648F") ,
                    ("Get-ViperBomb.ps1"               ,
"4771549A426A4E841A7D048613D65907BF7F416CF69797A1EAF9FAC8B28D797F") ,
                    ("Get-WhoisUtility.ps1"            ,
"A677D8026F18FBFF78C614CD3FC71BD6BE46EDC142D66CF9402EABAB9D988DCE") ,
                    ("Initialize-FeAdInstance.ps1"     ,
"D9D923D6919920866E905C3C710D1CD16F1A1ADBCF5952EF12CE71F54EEBEA79") ,
                    ("Install-BossMode.ps1"            ,
"25524DA6A44325BBFC5B4D4A863DE607B417CD4F3F57666627ECD9CB295AA07A") ,
                    ("Install-IISServer.ps1"           ,
"48F53BF8A3ECD087E7F395AA19F86D32849CD4F14B599F2AC6F7330F083E0D6C") ,
                    ("Install-PSD.ps1"                 ,
"0E0513C6BA4D98D1786E8606ABD5F6198947ADD43757E14D8138650DAB8D367F") ,
                    ("Invoke-cimdb.ps1"                ,
"567E8955B7D0A51569C5066AAE758E304B384592F55DF0D0A1176A8906885B56") ,
                    ("New-Document.ps1"                ,
"074638E4D16636BE3172F6A24B6B4BA8CC180BBD5E7E4C2424F2489A9E684C72") ,
                    ("New-EnvironmentKey.ps1"          ,
"B2F51FA6AFCFD499DE96CFD7458E216832B36204BB542FDB41647105860 3D04C") ,
                    ("New-FEConsole.ps1"               ,
"7B67102B7ED9856310B52FC2FAFA7A691AB75649F98C58036E1DCBF3BD7892A2") ,
                    ("New-FEFormat.ps1"                ,
"C4BFF5D8FBAC5ADBB79FEF848CE64A3C333C351EC1F50AC02468FCC0341AAAF4") ,
                    ("New-FEInfrastructure.ps1"        ,
"04C48E828FEF3DDCC6B07D914D088AB471B6C768C10F2DD38FD230A5B0566F67") ,
                    ("New-MarkdownFile.ps1"            ,
"425D7B38B3E5EDB06A13704881E6911201C445E75044FE93228A6F4C41E8A497") ,
                    ("New-TranscriptionCollection.ps1" ,
"DEB5CA810E582819BB7A86C7860BE9C14AAA1B46F59A0FB2BBE414C67321F16B") ,
                    ("Search-WirelessNetwork.ps1"     ,
"614FFE3CDC091001E46CEEBAF69AC2FE8C22D517E9F97DD85CBA8B037EC890AA") ,
                    ("Set-AdminAccount.ps1"           ,
"D217F33EE0BC5A00543B0EB8E99CC795D0035D835F0A0A6A8DD7DEF1F85F30B8") ,
                    ("Set-ScreenResolution.ps1"       ,
"60EE87AE8A1ADE31C2530BF3EC8E4BC03221692E599750265CE807648F9583E9") ,
                    ("Show-ToastNotification.ps1"     ,
"661B9C815FF1BAEEE4400F65126741177D6F5D122161EF0093309A9067B8344E") ,
                    ("Start-TCPSession.ps1"           ,
"B374EE3086B0E77C73A3A4D1F16CF9A150A2C11DE6E6A2B88B2C99FD47D56EAF") ,
                    ("Update-PowerShell.ps1"          ,
"EEA4AEEC98B7049F7273CDDCF58FCBD1702DDDEE1EB3C11B71DD76D30879F662") ,
                    ("Write-Element.ps1"              ,
"1AF8C0392304F9FC965ACAA1605C385FE479CC344514C2D5A532AB5DF81FC2D2") ,
                    ("Write-Theme.ps1"                ,
"64CDC6B7BB63816306C7A8410681DFB90B9555C08915D367F210739321250330")
                }
                Graphic
                {
                    ("background.jpg"                 ,
"94FD6CB32F8FF9DD360B4F98CEAA046B9AFCD717DA532AFEF2E230C981DAFEB5") ,
                    ("banner.png"                     ,
"057AF2EC2B9EC35399D3475AE42505CDBCE314B9945EF7C7BCB91374A8116F37") ,
                    ("icon.ico"                       ,
"594DAAFF448F5306B8B46B8DB1B420C1EE53FFD55EC65D17E2D361830659E58E") ,
                    ("OEMbg.jpg"                      ,
"D4331207D471F799A520D5C7697E84421B0FA0F9B574737EF06FC95C92786A32") ,
                    ("OEMlogo.bmp"                    ,
"98BF79CAE27E85C77222564A3113C52D1E75BD6328398871873072F6B363D1A8") ,
                    ("PSDBackground.bmp"              ,
"05ABBABDC9F67A95D5A4AF466149681C2F5E8ECD68F11433D32F4C0D04446F7E") ,
                    ("sdplogo.png"                    ,
"87C2B016401CA3F8F8FAD5F629AFB3553C4762E14CD60792823D388F87E2B16C")
                }
            }

            Return $List
        }
        LoadManifest()
        {
            $Out = @( )

            # Collects all of the files and names
            ForEach ($Type in "Control","Function","Graphic")
            {
                ForEach ($Item in $This.GetManifestList($Type))
                {
                    $Out += $This.ManifestListItem($Out.Count,$Type,$Item[0],$Item[1])
                }
            }

            # Determines maximum name length
            $Max = ($Out.Name | Sort-Object Length)[-1]

            ForEach ($Type in "Control","Function","Graphic")
            {
                # Adds + selects specified folder object
                $This.LoadFolder($Type)
                $Folder = $This.GetFolder($Type)

                # Loads each file + hash
                ForEach ($File in $Out | ? Source -eq $Type)
                {
                    $This.LoadFile($Folder,$Max.Length,$File)
                }

                $This.Update(0,"
          ")
            }
        }
        LoadFolder([String]$Type)
        {
            # Selects the correct folder name
            $ID   = $This.GetFolderName($Type)

            # Instantiates the specified folder
            $Item = $This.ManifestFolder($This.Manifest.Output.Count,$Type,$This.Root.Resource,$ID)

            # Logs validation of its existence, and adds if it does not
            Switch ([UInt32]!!$Item)
            {
                0
                {
                    $This.Update(
0,"------------------------------------------------------------------------------------------------------ ")
                    $This.Update( 0,("[!] {0} : {1}" -f $Item.Type.PadLeft(8," "), $Item.Fullname))
                    $This.Update(
0,"------------------------------------------------------------------------------------------------------ ")
                    $This.Update( 0,"
                ")
                }
                1
                {
                    $This.Manifest.Output += $Item
                    $This.Update(
0,"------------------------------------------------------------------------------------------------------ ")
                    $This.Update( 0,("[+] {0} : {1}" -f $Item.Type.PadLeft(8," "), $Item.Fullname))
                    $This.Update(
0,"------------------------------------------------------------------------------------------------------ ")
                    $This.Update( 0,"
                ")
                }
            }
        }
        LoadFile([Object]$Folder,[UInt32]$Max,[Object]$File)
        {
            $ID   = $File.Name
            $Hash = $File.Hash

            # Adds a specified file + hash into a specified folder object
            If ($ID -in $Folder.Item.Name)
            {
                Throw "Item already added"
            }

            # Instantiates the specified file
            $Item   = $This.ManifestFile($Folder,$ID,$Hash)
            $Label  = $ID.PadRight($Max," ")

            # Logs validation of its existence, and adds if it does not
            Switch ([UInt32]($ID -notin $Folder.Item.Name))
            {
                0
                {
                    $This.Update(-1,"[!] $Label")
                }
                1
                {
                    $Folder.Add($Item)
                    $This.Update( 1,"[o] $Label | $Hash ")
                }
            }
        }
        [Object] File([String]$Type,[String]$Name)
        {
            Return $This.GetFolder($Type).Item | ? Name -eq $Name
        }
        [Object] File([UInt32]$Index,[String]$Name)
        {
            Return $This.GetFolder($Index).Item | ? Name -eq $Name
        }
        [Object] _Control([String]$Name)
        {
            Return $This.File("Control",$Name)
        }
        [Object] _Function([String]$Name)
        {
            Return $This.File("Function",$Name)
        }
        [Object] _Graphic([String]$Name)
        {
            Return $This.File("Graphic",$Name)
        }
        [Void] WriteAllLines([String]$Path,[Object]$Object)
        {
            [System.IO.File]::WriteAllLines($Path,$Object,[System.Text.UTF8Encoding]$False)
        }
        [Void] Refresh()
        {
            # // ____________________________________________
            # // | Tests all manifest (folder/file) entries |
            # // ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾

            ForEach ($Item in $This.Module.Root.List() | Sort-Object Index -Descending)
            {
                Switch ($Item.Name)
                {
                    Registry
                    {
                        $This.Registry.TestPath()
                        $This.Root.Registry.Exists = $This.Registry.Exists
                    }
                    Resource
                    {
                        $This.Root.Resource.TestPath()
                        $This.Manifest.Refresh() | Out-Null
                    }
                    Module
                    {
                        $This.Root.Module.TestPath()
                    }
                    File
                    {
                        $This.Root.File.TestPath()
                    }
                    Manifest
                    {
                        $This.Root.Manifest.TestPath()
                    }
                    Shortcut
                    {
                        $This.Root.Shortcut.TestPath()
                    }
                }
            }
        }
        InstallItem([Object]$Item)
        {
            $Item.TestPath()

            Switch ($Item.Exists)
            {
                0
                {
                    Switch ($Item.Name)
                    {
                        Resource
                        {
                            $Item.Create()

                            $List       = $This.Manifest.Output | % Item

                            $Max        = ($List.Name | Sort-Object Length)[-1]
                            $C          = $List.Count + $This.Manifest.Output.Count
                            $I          = -1

                            $This.Update(1,"[@] Resource : $($Item.Fullname) ")
                            $This.Update(1,"               ($C) [directories/files] ")

                            ForEach ($Section in $This.Manifest.Output)
                            {
                                $Section.TestPath()
                                If (!$Section.Exists)
                                {
                                    $I ++
                                    $Status = "{0:p}" -f ($I/$C)

                                    $Section.Create()

                                    $This.Update(
1,"------------------------------------------------------------------------------------------------------- ")
                                    $This.Update( 1,("[~] {0} : {1} [$Status] " -f $Section.Type.PadRight(9," "),
$Section.FullName))
                                    $This.Update(
1,"------------------------------------------------------------------------------------------------------- ")
                                    $This.Update( 0,"
                                ")
                                }

                                ForEach ($File in $Section.Item)
                                {
                                    $I ++
                                    $Status = "{0:p}" -f ($I/$C)

                                    Switch ($File.Exists)
                                    {
                                        0
                                        {
                                            $File.Create()
                                            $File.Download()
                                            $File.Write()
                                            $This.Update(1,("[+] {0} [$Status] " -f $File.Name.PadRight($Max.Length," ")))
                                        }

                                        1
                                        {
                                            $This.Update(0,("[!] {0} [$Status] " -f $File.Name.PadRight($Max.Length," ")))
                                        }
                                    }
                                }

                                $This.Update(0,"
                          ")
                            }
                        }
                        Registry
                        {
                            $This.Update(1,"[@] Registry : $($Item.Fullname) ")
                            $This.Update(0,"
                      ")

                            $Key = $This.Registry.KeyTemp($Item.Fullname)
                            $Key.Open()
                            $Key.Create()

                            ForEach ($X in 0..($This.Registry.Property.Count-1))
                            {
                                $Prop        = $This.Registry.Property[$X]
                                $Key.Add($Prop.Name,$Prop.Value)

                                $This.Update(1,"[~] Property : [$($Prop.Name)], Value : [$($Prop.Value)]")
                                $Item.Exists = 1
                            }

                            $Key.Dispose()
                            $Item.TestPath()
                            $This.Update(0,"
                      ")
                        }
                        Module
                        {
                            $Item.Create()
                            $Item.TestPath()

                            $This.Update(1,"[+] PSModule : $($Item.Fullname) ")
                        }
                        File
                        {
                            $Item.Create()
                            $This.WriteAllLines($Item.Fullname,$This.Psm())
                            $Item.TestPath()
                            $This.Update(1,"[+] *.psm1   : $($Item.Fullname) ")
                        }
                        Manifest
                        {
                            $Splat = $This.PSDParam()
                            New-ModuleManifest @Splat
                            $Item.TestPath()
                            $This.Update(1,"[+] *.psd1   : $($Item.Fullname) ")
                        }
                        Shortcut
                        {
                            $Com                 = New-Object -ComObject WScript.Shell
                            $Object              = $Com.CreateShortcut($Item.Fullname)
                            $Object.TargetPath   = "PowerShell"
                            $Object.Arguments    = "-NoExit -ExecutionPolicy Bypass -Command `"Get-FEModule -Mode 1`""
                            $Object.Description  = $This.Description
                            $Object.IconLocation = $This._Graphic("icon.ico").Fullname
                            $Object.Save()

                            $Bytes               = [System.IO.File]::ReadAllBytes($Item.Fullname)
                            $Bytes[0x15]         = $Bytes[0x15] -bor 0x20

                            [System.IO.File]::WriteAllBytes($Item.Fullname,$Bytes)

                            $Item.TestPath()
                            $This.Update(1,"[+] *.lnk    : $($Item.Fullname) ")
                        }
                    }
                }
                1
                {
                    Switch ($Item.Name)
                    {
                        Resource
                        {
                            $This.Update(-1,"[!] Resource : $($Item.Fullname) [exists]")
                        }
                        Registry
                        {
                            $This.Update(-1,"[!] Registry : $($Item.Fullname) [exists]")
                        }
                        Module
                        {
                            $This.Update(-1,"[!] PSModule : $($Item.Fullname) [exists]")
                        }
                        File
                        {
                            $This.Update(-1,"[!] *.psm1   : $($Item.Fullname) [exists]")
                        }
                        Manifest
                        {
                            $This.Update(-1,"[!] *.psd1   : $($Item.Fullname) [exists]")
                        }
                        Shortcut
                        {
                            $This.Update(-1,"[!] *.lnk    : $($Item.Fullname) exists")
                        }
                    }
                }
            }
        }
        [Void] Install()
        {
            $This.Write(2,"Installing [~] $($This.Label())")

            $Setting = [System.Net.ServicePointManager]::SecurityProtocol
                       [System.Net.ServicePointManager]::SecurityProtocol = 3072


$This.Update(0,"======================================================================================================= ")
            $This.InstallItem($This.Root.Resource)

$This.Update(0,"------------------------------------------------------------------------------------------------------- ")

            $This.InstallItem($This.Root.Registry)

$This.Update(0,"------------------------------------------------------------------------------------------------------- ")
            $This.InstallItem($This.Root.Module)
            $This.InstallItem($This.Root.File)
            $This.InstallItem($This.Root.Manifest)
            $This.InstallItem($This.Root.Shortcut)

$This.Update(0,"======================================================================================================= ")
```

```
            [System.Net.ServicePointManager]::SecurityProtocol = $Setting

            $This.Write(2,"Installed [+] $($This.Label())")
        }
        RemoveItem([Object]$Item)
        {
            $Item.TestPath()

            Switch ($Item.Exists)
            {
                0
                {
                    Switch ($Item.Name)
                    {
                        Resource
                        {
                            $This.Update(1,"[_] Resource : $($Item.Fullname) ")
                        }
                        Registry
                        {
                            $This.Update(0,"[_] Registry : $($Item.Fullname) ")

                        }
                        Module
                        {
                            $This.Update(0,"[_] PSModule : $($Item.Fullname) ")
                        }
                        File
                        {
                            $This.Update(0,"[_] *.psm1   : $($Item.Fullname) ")
                        }
                        Manifest
                        {
                            $This.Update(0,"[_] *.psd1   : $($Item.Fullname) ")
                        }
                        Shortcut
                        {
                            $This.Update(0,"[_] *.lnk    : $($Item.Fullname)")
                        }
                    }
                }
                1
                {
                    Switch ($Item.Name)
                    {
                        Resource
                        {
                            $List       = $This.Manifest.Refresh()

                            $Max        = ($List.Name | Sort-Object Length)[-1]
                            $C          = $List.Count
                            $I          = -1

                            $This.Update(1,"[_] Resource : $($Item.Fullname) ")
                            $This.Update(1,"               ($C) [directories/files] ")

                            ForEach ($Section in $This.Manifest.Output)
                            {
                                $I ++
                                $Status = "{0:p}" -f ($I/$C)


$This.Update(1,"------------------------------------------------------------------------------------------------------ ")
                                $This.Update(1,("[_] {0} : {1} [$Status] " -f $Section.Type.PadRight(9," "),
$Section.FullName))

$This.Update(1,"------------------------------------------------------------------------------------------------------ ")
                                $This.Update(0,"
                          ")

                                ForEach ($File in $Section.Item)
                                {
                                    $I ++
                                    $Status = "{0:p}" -f ($I/$C)

                                    $File.Remove()
                                    $This.Update($File.Exists,("[_] {0} [$Status] " -f $File.Name.PadRight($Max.Length,"
")))
                                }

                                $This.Update(0,"
                          ")
                                $Section.Remove()
                            }

                            $Item.Remove()
                        }
                        Registry
                        {
                            $Object         = $This.Registry

                            $This.Update(1,"[ ] Registry : $($Item.Fullname) ")
                            $This.Update(0,"
                      ")

                            $Key            = $This.Registry.KeyTemp($Object.Path)
                            $Key.Open()
                            $Key.Create()
                            $Key.Remove()

                            ForEach ($Property in $Object.Property)
                            {
                                $This.Update(1,"[ ] Property : [$($Property.Name)]")
                                $Property.Exists = 0
                            }

                            $Object.Exists   = 0
                            $Key.Dispose()
                            $Item.Remove()

                            $This.Update(0,"
                      ")

                        }
                        Module
                        {
                            $Item.Remove()
                            $This.Update(1,"[_] PSModule : $($Item.Fullname) ")
                        }
                        File
                        {
                            $Item.Remove()
                            $This.Update(1,"[_] *.psm1   : $($Item.Fullname)")
                        }
                        Manifest
                        {
                            $Item.Remove()
                            $This.Update(1,"[_] *.psd1   : $($Item.Fullname)")
                        }
                        Shortcut
                        {
                            $Item.Remove()
                            $This.Update(1,"[_] *.lnk    : $($Item.Fullname)")
                        }
                    }
                }
            }
        }
        [Void] Remove()
        {
            $This.Update(0,"Removing [~] $($This.Label())")
            $This.Write(1,$This.Console.Last().Status)


$This.Update(0,"========================================================================================================== ")
            $This.RemoveItem($This.Root.Shortcut)
            $This.RemoveItem($This.Root.Manifest)
            $This.RemoveItem($This.Root.File)
            $This.RemoveItem($This.Root.Module)

$This.Update(0,"---------------------------------------------------------------------------------------------------------- ")
            $This.RemoveItem($This.Root.Registry)
```

```
$This.Update(0,"------------------------------------------------------------------------------------------------------ ")
            $This.RemoveItem($This.Root.Resource)

$This.Update(0,"====================================================================================================== ")

            $This.Write(1,"Removed [+] $($This.Label())")
        }
        [String] Psm()
        {
            $F      = @( )
            $Member = @( )

            # // __________
            # // | Header |
            # // ----------

            $F += "# Downloaded from {0}" -f $This.Source
            $F += "# {0}" -f $This.Resource
            $F += "# {0}" -f $This.Version.ToString()
            $F += "# <Types>"
            $This.Binaries() | % { $F += "Add-Type -AssemblyName $_" }

            # // _____________
            # // | Functions |
            # // -------------

            $F += "# <Functions>"
            ForEach ($File in $This.GetFolder("Function").Item)
            {
                $Base = $File.Name -Replace ".ps1",""
                If ($Member.Count -eq 0)
                {
                    $Member += "Export-ModuleMember -Function $Base,"
                }
                ElseIf ($Member.Count -gt 0)
                {
                    $Member += "$Base,"
                }

                $F += "# <{0}/{1}>" -f $File.Type, $File.Name
                $F += "# {0}" -f $File.Fullname
                If (!$File.Content)
                {
                    $File.GetContent()
                }
                $F += $File.Content
                $F += "# </{0}/{1}>" -f $File.Type, $File.Name
            }
            $Member[-1] = $Member[-1].TrimEnd(",")

            $F     += "# </Functions>"
            $F     += ""
            $Member | % { $F += $_ }
            $F     += ""
            $F     += "Write-Theme -InputObject `"Module [+] [FightingEntropy(`$([char]960))][$($This.Version)]`" -Palette
2"

            Return $F -join "`n"
        }
        [String[]] Binaries()
        {
            $Out = "PresentationFramework",
            "System.Runtime.WindowsRuntime",
            "System.IO.Compression",
            "System.IO.Compression.Filesystem",
            "System.Windows.Forms"

            Return $Out
        }
        [Hashtable] PSDParam()
        {
            Return @{

                GUID                 = $This.GUID
                Path                 = $This.Root.Manifest
                ModuleVersion        = $This.Version
                Copyright            = $This.Copyright
                CompanyName          = $This.Company
                Author               = $This.Author
                Description          = $This.Description
                RootModule           = $This.Root.File
                RequiredAssemblies   = $This.Binaries()
            }
        }
        [Object] ValidateFile([Object]$File)
        {
            Return [ValidateFile]::New($File)
        }
        [Object[]] Validation()
        {
            Return $This.Manifest.Full() | % { $This.ValidateFile($_) }
        }
        Validate()
        {
            $xList = $This.Validation()
            $This.Validate($xList)
        }
        Validate([Object[]]$xList)
        {
            $This.Write(3,"Validation [~] Module manifest")
            $Ct   = $xList | ? Match -eq 0

            Switch ($Ct.Count)
            {
                {$_ -eq 0}
                {
                    $This.Write(3,"Validation [+] All files passed validation")
                }
                {$_ -ne 0}
                {
                    $This.Write(1,"Validation [!] ($($Ct.Count)) files failed validation")
                }
            }
        }
        [String] ToString()
        {
            Return "<FightingEntropy.Module.Installer>"
        }
    }

    [InstallController]::New($Mode)
}

$Module = FightingEntropy.Module -Mode 0

    # // ____________________________________________________________________________________
    # // | Note: (FightingEntropy.Module -Mode 1) loads without writing stuff to the screen |
    # // ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
<#                                                                                                            __________/
\____________________________________________________________________________________________________________/ Function
  Output /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯

    ____________________________________________________________________
    | Here is the output of the function above                         | [Visual Studio]
    |==================================================================|
    | 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) 2023 (mcc85s/mcc85sx/sdp). All rights reserved. |
    | Guid        : 75f64b43-3b02-46b1-b6a2-9e86cccf4811                |
    | Date        : 04/03/2023 18:53:49                                 |
    | Version     : 2023.4.0                                            |
    | OS          : <FightingEntropy.Module.OS>                         |
    | Root        : <FightingEntropy.Module.Root>                       |
    | Manifest    : <FightingEntropy.Module.Manifest>                   |
    | Registry    : <FightingEntropy.Module.RegistryKey>                |
    ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

    ___________________________________________________________________________________________
    | Suppose I'd like to see the current version of the module based on the script above...? |
    ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```



```
    PS Prompt:\> $Module.GetFEVersion()

    Version  Date                Guid
    -------  ----                ----
    2023.4.0 04/03/2023 18:53:49 75f64b43-3b02-46b1-b6a2-9e86cccf4811
                                                                                                               _________/
\_____________________________________________________________________________________________________________/ Example
  Signature /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯

    __________________________________________________________________________________________
    | Michael C. Cook Sr. | Security Engineer | Secure Digits Plus LLC | 2023-04-03 18:55:27 |
    ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
                                                                                                             ___________/
\___________________________________________________________________________________________________________/ Signature
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\

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