```
<#
     ____    ____________________________________________________________________________________________________
    //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
    \\__//¯¯¯  [FightingEntropy(π)][2024.1.0]: 2024-01-21 20:40:54                                            ___//¯¯\\
     ¯¯¯\\___________________________________________________________________________________________________//¯¯\\__//
         ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯

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

    ____________________________________________________________________________________________
    | https://github.com/mcc85s/FightingEntropy/blob/main/Version/2024.1.0/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 || Virtual Lab - TCP Session                  || https://youtu.be/09c-fFbEQrU \\
    \\ 03/20/23 || Virtual Lab - Desktop Deployment           || https://youtu.be/i2_fafoIx6I //
    // 01/31/23 || New-VmController [Flight Test v2.0] Part I || https://youtu.be/nqTOmNIilxw \\
    \\ 01/12/23 || Virtualization Lab - FEDCPromo             || https://youtu.be/9v7uJHF-cGQ //
    //__________||____________________________________________||______________________________\\
    ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
    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)][2024.1.0]                                         |
    |========================================================================|
    | Version   | Date                | Guid                                 |
    |-----------|---------------------|--------------------------------------|
    | 2024.1.0  | 01/21/2024 15:45:50 | 2a354137-91c8-49c3-92d0-ee6275dab2fc |
    | 2023.8.0  | 08/07/2023 20:52:08 | 4b564727-b84b-4033-a716-36d1c5e3e62d |
    | 2023.4.0  | 04/03/2023 18:53:49 | 75f64b43-3b02-46b1-b6a2-9e86cccf4811 |
    ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
    _________________
    | Prerequisites |
    ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
    1) A system running [Windows PowerShell] on:
       - [Windows 10/11]
       - [Windows Server 2016/2019/2022]

    2) [Execution Policy] must be set to [bypass]

    3) Must be running a [PowerShell] session with [administrative privileges]
    ________________
    | Installation |
    ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
    1) As of 2024.1.0, this following link will host a [String] to the function that installs the latest version.

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

    Currently, that [String] points to this [file]...

    _____________________________________________________________________________________________
    | https://github.com/mcc85s/FightingEntropy/blob/main/Version/2024.1.0/FightingEntropy.ps1 |
    ---------------------------------------------------------------------------------------------

    To invoke the command, use:

    ________________________________________________________________________________________________________________
    | irm https://github.com/mcc85s/FightingEntropy/blob/main/Version/2024.1.0/FightingEntropy.ps1?raw=true | iex |
    ----------------------------------------------------------------------------------------------------------------

    ...or just (copy + paste) the content of that [file] into a [PowerShell] console, and press <enter>

    2) Once the [module is loaded into memory], it will create a variable called $Module, then enter the following:

    ____________________________________________________________________________________________________
    | Operation  | Instructions       | Description                                                     |
    |============|====================|=================================================================|
    | Latest     | $Module.Latest()   | Installs from the latest archive, and updates outstanding files |
    | Install    | $Module.Install()  | Installs from the embedded module manifest                      |
    | Remove     | $Module.Remove()   | Removes all traces of the module, registry, files, etc.         |
    ----------------------------------------------------------------------------------------------------

    _____________________________________________________________________________________________________
    | 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 ConsoleEntry
    {
        [UInt32]         $Index
        [String]       $Elapsed
        [Int32]          $State
        [String]        $Status
        Hidden [String] $String
        ConsoleEntry([UInt32]$Index,[String]$Time,[Int32]$State,[String]$Status)
        {
            $This.Index   = $Index
            $This.Elapsed = $Time
            $This.State   = $State
            $This.Status  = $Status
            $This.String  = $This.ToString()
        }
        [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()
    {
        $This.Reset()
    }
    [String] Elapsed()
    {
        Return @(Switch ($This.End.Set)
        {
            0 { [Timespan]([DateTime]::Now-$This.Start.Time) }
            1 { [Timespan]($This.End.Time-$This.Start.Time) }
        })
    }
    [Object] ConsoleTime([String]$Name)
    {
        Return [ConsoleTime]::New($Name)
    }
    [Object] ConsoleEntry([UInt32]$Index,[String]$Time,[Int32]$State,[String]$Status)
    {
        Return [ConsoleEntry]::New($Index,$Time,$State,$Status)
    }
    [Object] Collection()
    {
        Return [System.Collections.ObjectModel.ObservableCollection[Object]]::New()
    }
    [Void] SetStatus()
    {
        $This.Status = $This.ConsoleEntry($This.Output.Count,
                                          $This.Elapsed(),
                                          $This.Status.State,
                                          $This.Status.Status)
    }
    [Void] SetStatus([Int32]$State,[String]$Status)
    {
        $This.Status = $This.ConsoleEntry($This.Output.Count,
                                          $This.Elapsed(),
                                          $State,
                                          $Status)
    }
    Initialize()
    {
        If ($This.Start.Set -eq 1)
        {
            $This.Update(-1,"Start [!] Error: Already initialized, try a different operation or reset.")
        }
        $This.Start.Toggle()
        $This.Update(0,"Running [~] ($($This.Start))")
    }
    Finalize()
    {
        If ($This.End.Set -eq 1)
        {
            $This.Update(-1,"End [!] Error: Already initialized, try a different operation or reset.")
        }
        $This.End.Toggle()
        $This.Span = $This.Elapsed()
        $This.Update(100,"Complete [+] ($($This.End)), Total: ($($This.Span))")
    }
    Reset()
    {
        $This.Start  = $This.ConsoleTime("Start")
        $This.End    = $This.ConsoleTime("End")
        $This.Span   = $Null
        $This.Status = $Null
        $This.Output = $This.Collection()
    }
    Write()
    {
        $This.Output.Add($This.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 $This.Output | % ToString
    }
    [String] ToString()
    {
        If (!$This.Span)
        {
            Return $This.Elapsed()
        }
        Else
        {
            Return $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 "<FEModule.Theme.Block>"
    }
}

# // ==============================================
# // | 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 "<FEModule.Theme.Track>"
    }
}

# // ============================================
# // | 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 "<FEModule.Theme.Stack>"
    }
}

# // ===================================================
# // | 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 "<FEModule.OS.Property>"
    }
}

# // ==========================================================
# // | 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()
    {
        Return "<FEModule.OS.Property.Set>"
    }
}

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

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

        Else
        {
            $Item = $This.GetWinType()
        }

        Return $Item
    }
    [String] ToString()
    {
        Return "<FEModule.OS.Controller>"
    }
}

# // ======================================
# // | Enumerates the manifest item types |
# // ======================================

Enum ManifestSectionType
{
    Control
    Function
    Graphic
}

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

Class ManifestSection
{
    [UInt32]  $Index
    [String] $Source
    [String]   $Name
    [String]   $Hash
    ManifestSection([UInt32]$Index,[String]$Source,[String]$Name,[String]$Hash)
    {
        $This.Index  = $Index
        $This.Source = $Source
        $This.Name   = $Name
        $This.Hash   = $Hash
    }
    [String] ToString()
    {
        Return "<FEModule.Manifest.Section>"
    }
}

# // ==============================================================
# // | 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()
    {
        $X        = 0
        $xContent = $Null
        Do
        {
            Try
            {
                $xContent = Invoke-WebRequest $This.Source -UseBasicParsing -TimeoutSec 5 | % Content
                $X ++
            }
            Catch
            {

            }
        }
        Until (!!$xContent -or $X -eq 5)

        If (!$xContent)
        {
            Throw "Exception [!] File {0} failed to download" -f $This.Name
        }

        Switch -Regex ($This.Name)
        {
            "\.+(jpg|jpeg|png|bmp|ico)"
            {
                $This.Content = $xContent
            }
            "\.+(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"
            }
            Default
            {
                $This.Content = $xContent
            }
        }
    }
    Write()
    {
        If (!$This.Content)
        {
            Throw "Exception [!] Content not assigned, cannot (write/set) content."
        }

        If (!$This.Exists)
        {
            $This.Create()
        }

        Try
        {
            Switch -Regex ($This.Name)
            {
                "\.+(jpg|jpeg|png|bmp|ico)"
                {
                    [System.IO.File]::WriteAllBytes($This.Fullname,[Byte[]]$This.Content)
                }
                "\.+(xml|txt|cs)"
                {
                    [System.IO.File]::WriteAllText($This.Fullname,$This.Content)
                }
                Default
                {
                    [System.IO.File]::WriteAllText($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
        {
            Switch -Regex ($This.Name)
            {
                "\.+(jpg|jpeg|png|bmp|ico)"
                {
                    [System.IO.File]::ReadAllBytes($This.Fullname)
                }
                "\.+(xml|txt|cs)"
                {
                    [System.IO.File]::ReadAllText($This.Fullname,[System.Text.UTF8Encoding]$False)
                }
                Default
                {
                    [System.IO.File]::ReadAllLines($This.Fullname,[System.Text.UTF8Encoding]$False)
                }
            }
        }
        Catch
        {
            Throw "Exception [!] An unspecified error has occurred"
        }
    }
    [String] ToString()
    {
        Return "<FEModule.Manifest.File>"
    }
}

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

Class ManifestFolder
{
    Hidden [UInt32]    $Index
    Hidden [UInt32]     $Mode
    [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 "<FEModule.Manifest.Folder>"
    }
}

# // ====================================================================
# // | 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()
```

```powershell
            $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 $D
    }
    Validate()
    {
        ForEach ($Folder in $This.Output)
        {
            $Folder.Exists = [System.IO.Directory]::Exists($Folder.Fullname)
            If ($Folder.Exists)
            {
                ForEach ($File in $Folder.Item)
                {
                    $File.Exists = [System.IO.File]::Exists($File.Fullname)
                    If ($File.Exists)
                    {
                        $File.GetContent()
                    }
                }
            }
        }
    }
    [String] ToString()
    {
        Return "<FEModule.Manifest.Controller>"
    }
}

# // ═══════════════════════════════════
# // | 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
    }
    [String] ToString()
    {
        Return "<FEModule.Registry.Template>"
    }
}

# // ===========================================
# // | Works as a PowerShell Registry provider |
# // ===========================================

Class RegistryTemporaryKey
{
    Hidden [Microsoft.Win32.RegistryKey]    $Key
    Hidden [Microsoft.Win32.RegistryKey] $Subkey
    [String]                              $Enum
    [String]                              $Hive
    [String]                              $Path
    [String]                              $Name
    Hidden [String]                   $Fullname
    RegistryTemporaryKey([String]$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()
        }
    }
    [String] ToString()
    {
        Return "<FEModule.Registry.Temporary.Key>"
    }
}

# // ========================================================
# // | 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 "<FEModule.Registry.Key.Property>"
    }
}

# // ===========================================================
# // | 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            = $This.RegistryTemporaryKey($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             = $This.RegistryTemporaryKey($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,$Property)
    }
    [Object] TemporaryKey([String]$Path)
    {
        Return [RegistryTemporaryKey]::New($Path)
    }
    [String] ToString()
    {
        Return "<FEModule.Registry.Key>"
    }
}

# // ==================================================
# // | 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 RootController
{
    Hidden [String] $Name
    [Object]    $Registry
    [Object]    $Resource
    [Object]      $Module
    [Object]        $File
    [Object]    $Manifest
    [Object]    $Shortcut
    RootController([String]$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 "<FEModule.Root.Controller>"
    }
}

# // ==========================================
# // | 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{1,}\.\d{1,}" }
            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($Path))
        {
            [System.IO.File]::Create($Path).Dispose()
        }

        Return Get-FileHash $Path | % Hash
    }
    [String] ToString()
    {
        Return "<FEModule.Validate.File>"
    }
}

# // ===============================================================
# // | Specifically meant to categorize available version archives |
# // ===============================================================

Class MarkdownArchiveEntry
{
    Hidden [DateTime]   $Real
    [String]            $Date
    [String]            $Name
    [String]            $Link
    Hidden [String] $NameLink
    [String]            $Hash
    MarkdownArchiveEntry([String]$Date,[String]$Name,[String]$Hash,[String]$Link)
    {
        $This.Date     = $Date
        $This.Real     = [DateTime]$This.Date
        $This.Name     = $Name
        $This.Link     = $Link
        $This.NameLink = "[**{0}**]({1})" -f $This.Name,$This.Link
        $This.Hash     = $Hash
    }
    MarkdownArchiveEntry([String]$Line)
    {
        $This.Date     = [Regex]::Matches($Line,"\d{4}\-\d{2}\-\d{2} \d{2}\:\d{2}\:\d{2}").Value
        $This.Real     = [DateTime]$This.Date
        $This.Name     = [Regex]::Matches($Line,"\*\*\d{4}\-\d{2}\-\d{2}_\d{6}.zip\*\*").Value.Trim("*")
        $This.Link     = [Regex]::Matches($Line,"https.+.zip").Value
```

```powershell
        $This.NameLink = "[**{0}**]({1})" -f $This.Name,$This.Link
        $This.Hash     = [Regex]::Matches($Line,"[A-F0-9]{64}").Value
    }
    [String] Prop([String]$Property,[String]$Char)
    {
        $Prop = $This.$Property
        Return $Prop.PadRight($Prop.Length,$Char)
    }
    [String[]] GetOutput()
    {
        Return "| {0} | {1} | {2} |" -f $This.Prop("Date"," "),
                                        $This.Prop("NameLink"," "),
                                        $This.Prop("Hash"," ")
    }
    [String] ToString()
    {
        Return "<FEModule.Markdown.Archive.Entry>"
    }
}


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

Class ModuleController
{
    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) 2024 (mcc85s/mcc85sx/sdp). All rights reserved."
    [Guid]             $Guid = "2a354137-91c8-49c3-92d0-ee6275dab2fc"
    [DateTime]         $Date = "01/21/2024 15:45:50"
    [Version]       $Version = "2024.1.0"
    [Object]             $OS
    [Object]           $Root
    [Object]       $Manifest
    [Object]       $Registry
    ModuleController([Switch]$Flags)
    {
        $This.Mode = 0
        $This.Main()
    }
    ModuleController()
    {
        $This.Mode = 0
        $This.Main()
    }
    ModuleController([UInt32]$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," ".PadLeft(102," "))

        # 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] Now()
{
    Return [DateTime]::Now.ToString("yyyy-MMdd_HHmmss")
}
[String] ProgramData()
{
    Return [Environment]::GetEnvironmentVariable("ProgramData")
}
[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
```

```powershell
    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" | ? { $_ -match "^\|\s\*\*\d{4}\.\d{1,}\.\d{1,}\*\*" } | % { [FEVersion]$_ }
}
[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
        {
            [OSController]::New()
        }
        Root
        {
            [RootController]::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 $xName
}
[Object] ManifestSection([UInt32]$Index,[String]$Source,[String]$Name,[String]$Hash)
{
    Return [ManifestSection]::New($Index,$Source,$Name,$Hash)
}
[Object[]] GetManifestList([String]$Name)
{
    $List = Switch ($Name)
    {
        Control
        {
            ("Computer.png"                    , "87EAB4F74B38494A960BEBF69E472AB0764C3C7E782A3F74111F993EA31D1075") ,
            ("DefaultApps.xml"                 , "EEC0F0DFEAC1B4172880C9094E997C8A5C5507237EB70A241195D7F16B06B035") ,
            ("down.png"                        , "0F14F2184720CC89911DD0FB234954D83275672D5DBA3F48CBDAFA070C0376B4") ,
            ("failure.png"                     , "59D479A0277CFFDD57AD8B9733912EE1F3095404D65AB630F4638FA1F40D4E99") ,
            ("FEClientMod.xml"                 , "326C8D3852895A3135144ACCBB4715D2AE49101DCE9E64CA6C44D62BD4F33D02") ,
            ("FEServerMod.xml"                 , "3EA9AF3FFFB5812A3D3D42E5164A58EF2FC744509F2C799CE7ED6D0B0FF9016D") ,
            ("header-image.png"                , "38F1E2D061218D31555F35C729197A32C9190999EF548BF98A2E2C2217BBCB88") ,
            ("left.png"                        , "BE62B17A91BDCC936122557397BD90AA3D81F56DDA43D62B5FDBCEDD10C7AFFB") ,
            ("MDTClientMod.xml"                , "B2BA25AEB67866D17D8B22BFD31281AFFF0FFE1A7FE921A97C51E83BF46F8603") ,
            ("MDTServerMod.xml"                , "C4B12E67357B54563AB042617CEC2B56128FD03A9C029D913BB2B6CC65802189") ,
            ("MDT_LanguageUI.xml"              , "8968A07D56B4B2A56F15C07FC556432430CB1600B8B6BBB13C332495DEE95503") ,
            ("PSDClientMod.xml"                , "C90146EECF2696539ACFDE5C2E08CFD97548E639ED7B1340A650C27F749AC9CE") ,
            ("PSDServerMod.xml"                , "C90146EECF2696539ACFDE5C2E08CFD97548E639ED7B1340A650C27F749AC9CE") ,
            ("right.png"                       , "A596F8859E138FA362A87E3253F64116368C275CEE0DA3FDD6A686CBE7C7069A") ,
            ("success.png"                     , "46757AB0E2D3FFFFDBA93558A34AC8E36F972B6F33D00C4ADFB912AE1F6D6CE2") ,
            ("up.png"                          , "09319D3535B26451D5B7A7F5F6F6897431EBDC6AED261288F13C2C65D50C4346") ,
            ("vendorlist.txt"                  , "A37B6652014467A149AC6277D086B4EEE7580DDB548F81B0B2AA7AC78C240874") ,
            ("warning.png"                     , "CC05A590DE7AD32AEB47E117AA2DD845F710080F9A3856FBCDC9BC68106C562F") ,
            ("Wifi.cs"                         , "653A421E4F29882DA8276F9D543FD792D249BE141F2043BDC65C17C6B6FAC77B") ,
            ("zipcode.txt"                     , "E471E887F537FA295A070AB41E21DEE978181A92CB204CA1080C6DC32CBBE0D8")
        }
        Function
        {
            ("Copy-FileStream.ps1"             , "862B3E6913475FC321387FAAE8C0BA3298759D7F55D7E11D2FDDF6E34257BECC") ,
            ("Get-AssemblyList.ps1"            , "EBEF2B109FE5646522579BDBBC6BE7BD7465C0CA5D10405248A13C9495FA40E4") ,
            ("Get-ControlExtension.ps1"        , "A7ABC20AA24A13DDFBE38DA83CB1DC52032504C60A6EAA055816DCDE94B01966") ,
            ("Get-DcomSecurity.ps1"            , "8507E507DECF99A078C45C3157F27D93DE35B0004F4C54DFBFF5ACB4559462A3") ,
            ("Get-EnvironmentKey.ps1"          , "AB1B926D0B567F9ED943D83C58BC0274129D9D0D2BFE7EAADEEBD99A6EAA448E") ,
            ("Get-EventLogArchive.ps1"         , "DFD1FF7AB141951938A931F3FDDFB275DC72C1151E02B2BFC3303080154E4995") ,
            ("Get-EventLogConfigExtension.ps1" , "48130EED8EED86A2B365912FF7BD440DE2310759159AE8EFCD8B03809C92BB5A") ,
            ("Get-EventLogController.ps1"      , "644BDF1ECBC6BF4A0E9D611D0F8C30115019D996CCB07184FAADEF30A73EFEB8") ,
            ("Get-EventLogProject.ps1"         , "29AEA45483422269 7F83400888FE74EEB77B6ABF43707D844AD5A7E77B24E3CB") ,
            ("Get-EventLogRecordExtension.ps1" , "D0A6C8AD8801060EF0EE7CDF39065321E16B233E9755E714DB0C030AC95BF9A0") ,
            ("Get-EventLogXaml.ps1"            , "CD667980014974ABC7287678E19C3959CE87660C09DBF2EBB96D18B962C3D390") ,
            ("Get-FEADLogin.ps1"               , "C900FE37D5FC0F63A1E0BC5DD9B36C57448331A8A479C2E0A31880E8D9E35CF4") ,
            ("Get-FEDCPromo.ps1"               , "4F668EE8E56F9E8C74D5C015411C439DDC54978B55D0CEB6786D7412098A47CB") ,
            ("Get-FEDevice.ps1"                , "409D7C7F190FCD690A6618B542C0352B6D682D2C7DE0A62973A2B9CB6266F98F") ,
            ("Get-FEImageManifest.ps1"         , "F01DF0E164A47A56E2F9D9F4CD2F93F3C703B9AAA7C1C575013062318 7BE1D5E") ,
            ("Get-FEModule.ps1"                , "B77F937710C1CA67E6A9B7A15014ADAD6D6A9DFEA2F2CCB5A930990AAEB2476E") ,
            ("Get-FENetwork.ps1"               , "874C435C5AFB476FCFA707FEEDEAB03AEA1526B40AAD5F8D78C00181E08093F2") ,
            ("Get-FEProcess.ps1"               , "0D8AA28C157D001A5A1222DA72076C168075CC431BE0C4C88FA64434B96FB29C") ,
            ("Get-FESystem.ps1"                , "45125620B1AB92BD84FCC54BB823C35BADA82092BA08B835D1E5F68ECEDBCAA0") ,
            ("Get-MdtModule.ps1"               , "F4B9015A37930052ACDF583C8A35A22FF5C6F545720E2F888D671ADA811E79E7") ,
            ("Get-PowerShell.ps1"              , "8E566FA8AD0C23919501012AA7266691729D327F83D6C0792E4539EB583CA041") ,
            ("Get-PropertyItem.ps1"            , "92CF80AB4DD5115E333E1CE67F9E24DB7701FC0DEB15F16E11C06667325E5CD1") ,
            ("Get-PropertyObject.ps1"          , "5F722AE1FAA35C89D6588768B106344B193D2500474E5186BC9B8D22A3130B52") ,
            ("Get-PsdLog.ps1"                  , "6411411A6B660F72E872DDE58503039180380C39014983E51CE4D1DC40EE2882") ,
            ("Get-PsdLogGUI.ps1"               , "468EC4816E873926BE27A1F8432131F360816DB0A0BBDD7E3773E5EAD061DF8C") ,
            ("Get-PsdModule.ps1"               , "7CBDE4526EC57758002D00C6D8BE50C5E4E7292351C1E4ED2658224C40C407E7") ,
            ("Get-ThreadController.ps1"        , "2E731F4282F6CA2281E168E8DB6C7E6ED3811AA6F15347C10581A943DB2117B5") ,
            ("Get-UserProfile.ps1"             , "10E3A87935D90E61F0030011D4BEE99877E9B432A4B507EFE8577C87AEC2BE69") ,
            ("Get-ViperBomb.ps1"               , "58C1491DE7B8C9FD243462BA1041BC3AE08330C43B44DA3DB7B8727B83795BDF") ,
            ("Get-WhoisUtility.ps1"            , "CFFCA2A3C03293F9119B9BFEC3A99E8C4902999F66480D7D1617D2E3D2359C50") ,
            ("Initialize-FeAdInstance.ps1"     , "68064EBEF39724EF82FCCA7150063463265BBE0C2DDC1BFFE8497B884C810266") ,
            ("Initialize-VmNode.ps1"           , "88C0A4F16881E77A2C52CC48DEFC145FD1A9E0DB3B96ADDBB7105160046826CD") ,
            ("Install-IISServer.ps1"           , "C8C0EA6332560E3BCF0B37FBDF45436D54A65ED005705BE29AA25F18B33ABA54") ,
            ("Install-Psd.ps1"                 , "7CF53D11B15CF7E712A8E35142094C4563A9DCD08917C65D2022C7B014BE4E9F") ,
            ("Invoke-cimdb.ps1"                , "8835574220B607F27C45A831CD5CECBD6757364486AF9508DF71FC9495B82D0B") ,
            ("New-Document.ps1"                , "342AE1373890D6036AECF2A53D93F3A2C67E0CE3A951E002BDA117FEBF4C62FC") ,
            ("New-EnvironmentKey.ps1"          , "18A1BEBD461E666AAB42383B8C4ECA950929552B1C9704B53CBF6FF002936FFC") ,
            ("New-FEConsole.ps1"               , "89412440E1C2A65D7F33A7A93CAEC8B26C6E2E2A9E41E1DE320A401C87A7F871") ,
            ("New-FEFormat.ps1"                , "95126B932F16DB2634446B83372948F6538066D6B3A130D09D604AD315752099") ,
            ("New-FEInfrastructure.ps1"        , "D93A297BF83BEB130B9F9D24E855654F8FB670A594AC4AF8BC338C7CA6521F24") ,
            ("New-MarkdownFile.ps1"            , "5A3D759D55390C4F72AFC546C977E69F0F9BE5AF2A45D96010E8550B0CF27C2B") ,
            ("New-TranscriptionCollection.ps1" , "BC3B020A6F0CF8CD5CF8C06CF2EE725A7E3C2CC2886F471CB180693603 2D4307") ,
            ("New-VmController.ps1"            , "6BB7336DD41EFA67808C8E4D335F072AF114F77E459B48743C291DCB3A1C14ED") ,
            ("Search-WirelessNetwork.ps1"      , "30A3024E8FCFAFC93B953CE44CC1E03FA901313063F2950020785 4E8F0E856D2") ,
            ("Set-AdminAccount.ps1"            , "C5E6A661A7DEF8B8C791DE1AED278586B2709A0C6A550FFF690FF707464DF732") ,
            ("Set-ScreenResolution.ps1"        , "9F14E7E9190ABD299F7A21F1E7A57809EBF0E5182099DE845573ABB2E55BDFCF") ,
            ("Show-ToastNotification.ps1"      , "61BDDF6AF8143CEA43FA1648F2AF172D68A1CCE4750D326449EB50A742EAC04F") ,
            ("Start-TCPSession.ps1"            , "878BA5EF733666431D5EC94C2C6C132B6E4F4F6DFA1664AE872F7F0F7FCD59CE") ,
            ("Update-PowerShell.ps1"           , "BA12BE91B23691DE30CCF7583CCFA397B56B7E9E8B89B157C9A79FC808F1F0C5") ,
            ("Write-Element.ps1"               , "D30BCDDD5352D70C730B70E458D4900CE7904EEDF9A387B29EA4F69EA3D16327") ,
            ("Write-Theme.ps1"                 , "1FC13440093B76ABADBD6960FBE788F5029FF288E8B3ABE95781994FD14935BB") ,
            ("Write-Xaml.ps1"                  , "33D7A14875469A67EB1DFEE2805DA27E734788A3CD001A45FAE46B6C7BDDC7CF")
        }
        Graphic
        {
            ("background.jpg"                  , "94FD6CB32F8FF9DD360B4F98CEAA046B9AFCD717DA532AFEF2E230C981DAFEB5") ,
            ("banner.png"                      , "057AF2EC2B9EC35399D3475AE42505CDBCE314B9945EF7C7BCB91374A8116F37") ,
            ("icon.ico"                        , "594DAAFF448F5306B8B46B8DB1B420C1EE53FFD55EC65D17E2D361830659E58E") ,
            ("OEMbg.jpg"                       , "D4331207D471F799A520D5C7697E84421B0FA0F9B574737EF06FC95C92786A32") ,
            ("OEMlogo.bmp"                     , "98BF79CAE27E85C77222564A3113C52D1E75BD6328398871873072F6B363D1A8") ,
            ("PSDBackground.bmp"               , "05ABBABDC9F67A95D5A4AF466149681C2F5E8ECD68F11433D32F4C0D04446F7E") ,
            ("sdplogo.png"                     , "87C2B016401CA3F8F8FAD5F629AFB3553C4762E14CD60792823D388F87E2B16C")
        }
    }

    Return $List
}
[String[]] ManifestEnum()
{
    Return [System.Enum]::GetNames([ManifestSectionType])
}
LoadManifest()
{
    $Out = @( )

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

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

    ForEach ($Type in $This.ManifestEnum())
    {
        # 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)
        }

        $This.Update(0," ".PadLeft(102," "))
    }
}
LoadFolder([String]$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,"-".PadLeft(102,"-"))
            $This.Update( 0,("[!] {0} : {1}" -f $Item.Type.PadLeft(8," "), $Item.Fullname))
            $This.Update( 0,"-".PadLeft(102,"-"))
            $This.Update( 0," ".PadLeft(102," "))
        }
        1
        {
            $This.Manifest.Output += $Item
            $This.Update( 0,"-".PadLeft(102,"-"))
            $This.Update( 0,("[+] {0} : {1}" -f $Item.Type.PadLeft(8," "), $Item.Fullname))
            $This.Update( 0,"-".PadLeft(102,"-"))
            $This.Update( 0," ".PadLeft(102," "))
        }
    }
}
LoadFile([Object]$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 ($Sx in $This.Manifest.Output)
                    {
                        $Sx.TestPath()
                        If (!$Sx.Exists)
                        {
                            $I ++
                            $St = "{0:p}" -f ($I/$C)

                            $Sx.Create()

                            $This.Update( 1,"-".PadLeft(102,"-"))
                            $This.Update( 1,("[~] {0} : {1} [$St] " -f $Sx.Type.PadRight(9," "), $Sx.FullName))
                            $This.Update( 1,"-".PadLeft(102,"-"))
                            $This.Update( 0," ".PadLeft(102," "))
                        }

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

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

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

                        $This.Update(0," ".PadLeft(102," "))
                    }
                }
                Registry
                {
                    $This.Update(1,"[@] Registry : $($Item.Fullname) ")
                    $This.Update(0," ".PadLeft(102," "))

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

                    $Max = @{

                        Name = ($This.Registry.Property.Name | Sort-Object Length)[-1].Length
                    }

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

                        $This.Update(1,"[+] $($Prop.Name.PadRight($Max.Name," ")) : $($Prop.Value)")
                        $Item.Exists = 1
                    }

                    $Key.Dispose()
                    $Item.TestPath()
                    $This.Update(0," ".PadLeft(102," "))
                }
                Module
                {
                    $Item.Create()

                    $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
```

```powershell
                        $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,"=".PadLeft(102,"="))
        $This.InstallItem($This.Root.Resource)
        $This.Update(0,"-".PadLeft(102,"-"))

        $This.InstallItem($This.Root.Registry)
        $This.Update(0,"-".PadLeft(102,"-"))
        $This.InstallItem($This.Root.Module)
        $This.InstallItem($This.Root.File)
        $This.InstallItem($This.Root.Manifest)
        $This.InstallItem($This.Root.Shortcut)
        $This.Update(0,"=".PadLeft(102,"="))

        [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 ($Sx in $This.Manifest.Output)
            {
                $I ++
                $St = "{0:p}" -f ($I/$C)

                $This.Update(1,"-".PadLeft(102,"-"))
                $This.Update(1,("[_] {0} : {1} [$St] " -f $Sx.Type.PadRight(9," "), $Sx.FullName))
                $This.Update(1,"-".PadLeft(102,"-"))
                $This.Update(0," ".PadLeft(102," "))

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

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

                $This.Update(0," ".PadLeft(102," "))
                $Sx.Remove()
            }

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

            $This.Update(1,"[ ] Registry : $($Item.Fullname) ")
            $This.Update(0," ".PadLeft(102," "))

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

            $Max = @{

                Name = ($This.Registry.Property.Name | Sort-Object Length)[-1].Length
            }

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

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

            $This.Update(0," ".PadLeft(102," "))

        }
        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)

        $This.Update(0,"=".PadLeft(102,"="))
        ForEach ($Item in "Shortcut","Manifest","File","Module")
        {
            $This.RemoveItem($This.Root.$Item)
        }
        $This.Update(0,"-".PadLeft(102,"-"))
        $This.RemoveItem($This.Root.Registry)
        $This.Update(0,"-".PadLeft(102,"-"))
        $This.RemoveItem($This.Root.Resource)
        $This.Update(0,"=".PadLeft(102,"="))

        $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()
    }
}
Latest()
{
    $This.Write(2,"Installing [~] $($This.Label())")

    If (![System.IO.Directory]::Exists($This.Root.Resource))
    {
        $This.Root.Resource.Create()
    }

    $String    = "{0}/blob/main/Version/{1}/readme.md?raw=true" -f $This.Source, $This.Version.ToString()
    $Content   = (Invoke-RestMethod $String).Split("`n")
    $List      = @( )

    ForEach ($Line in $Content)
    {
        If ($Line -match "https.+\.zip")
        {
            $List += $This.ArchiveEntry($Line)
        }
    }

    If ($List.Count -eq 0)
    {
        Throw "[!] No archive available, use Install()"
    }

    $Item      = ($List | Sort-Object Real)[-1]

    $This.Update(0,"====[Downloading Latest Archive]====".PadRight(102,"="))
    $This.Update(0,"")
    $This.Update(0,"    Date : $($Item.Date)")
    $This.Update(0,"    Name : $($Item.Name)")
    $This.Update(0,"    Link : $($Item.Link)")
    $This.Update(0,"    Hash : $($Item.Hash)")
    $This.Update(0,"")

    $Src       = "{0}?raw=true" -f $Item.Link
    $Target    = "{0}\{1}" -f $This.Root.Resource.Fullname, $Item.Name

    Start-BitsTransfer -Source $Src -Destination $Target

    $Hash      = Get-FileHash $Target | % Hash
    If ($Item.Hash -notmatch $Hash)
    {
        $This.Update(-1,"Error       [!] Invalid hash")
        [System.IO.File]::Delete($Target)
        Throw $This.Console.Status
    }

    Expand-Archive $Target -DestinationPath $This.Root.Resource -Force
    [System.IO.File]::Delete($Target)
    $This.Manifest.Validate()

    $This.Update(0,"=".PadLeft(102,"="))
    $This.Update(0,"[@] Resource : $($This.Root.Resource)")
    $Ct = $This.Manifest | % { $_.Output.Count + $_.Full().Count }
    $This.Update(0,"               ($Ct) [directories/files]")
    ForEach ($Folder in $This.Manifest.Output)
    {
        $This.Update(0,"-".PadLeft(102,"-"))
        $This.Update(0,("[~] {0} : {1}" -f $Folder.Type.PadRight(9," "), $Folder.Fullname))
        $This.Update(0,"-".PadLeft(102,"-"))
        $This.Update(0," ".PadLeft(102," "))
        ForEach ($File in $Folder.Item)
        {
            $This.Update(0,"[+] $($File.Name)")
        }

        $This.Update(0," ".PadLeft(102," "))
    }

    $This.Update(0,"-".PadLeft(102,"-"))

    If ($This.Root.Registry.Exists -eq 0)
    {
        $This.InstallItem($This.Root.Registry)
    }

    $This.Update(0,"-".PadLeft(102,"-"))

    $This.UpdateManifest()

    $This.Update(0,"=".PadLeft(102,"="))
    $This.Write(2,"Installed [+] $($This.Label())")
}
UpdateManifest()
{
    $List = $This.Validation()
    $Pull = $List | ? Match -eq 0

    If ($Pull.Count -ne 0)
    {
        ForEach ($ID in "Shortcut","Manifest","File","Module")
        {
            $Item = $This.Root.$ID
            If ($Item.Exists)
            {
                $This.RemoveItem($Item)
            }
        }

        ForEach ($File in $Pull)
        {
            $Folder = $This.Manifest.Output | ? Type -eq $File.Type
            $Item   = $Folder.Item | ? Name -eq $File.Name
            $Item.Download()
            $Item.Write()
            $Item.Exists = 1
        }

        ForEach ($Item in "Module","File","Manifest","Shortcut")
        {
            $This.InstallItem($This.Root.$Item)
        }
    }
}
[Object] ArchiveEntry([String]$Line)
{
    Return [MarkdownArchiveEntry]::New($Line)
}
[Object] ValidateFile([Object]$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] DateTime()
```

```
        {
            Return [DateTime]::Now.ToString("yyyy-MM-dd HH:mm:ss")
        }
        [String] ToString()
        {
            Return "<FEModule.Module.Controller>"
        }
    }

    [ModuleController]::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                          |
    |===================================================================|
    | 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) 2024 (mcc85s/mcc85sx/sdp). All rights reserved. |
    | Guid        : 2a354137-91c8-49c3-92d0-ee6275dab2fc                |
    | Date        : 01/21/2024 15:45:50                                 |
    | Version     : 2024.1.0                                            |
    | OS          : <FEModule.OS.Controller>                            |
    | Root        : <FEModule.Root.Controller>                          |
    | Manifest    : <FEModule.Manifest.Controller>                      |
    | Registry    : <FEModule.Registry.Key>                             |
    --------------------------------------------------------------------

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

    PS Prompt:\> $Module.GetFEVersion()

    Version  Date                Guid
    -------  ----                ----
    2024.1.0 01/21/2024 15:45:50 2a354137-91c8-49c3-92d0-ee6275dab2fc
                                                                                                             _________/
\___________________________________________________________________________________________________________/ Example
  Signature /---------------------------------------------------------------------------------------------------------\
/-----------

    _________________________________________________________________________________________
    | Michael C. Cook Sr. | Security Engineer | Secure Digits Plus LLC | 2024-01-21 20:40:54 |
    -----------------------------------------------------------------------------------------

                                                                                                           ___________/
\_________________________________________________________________________________________________________/ Signature
/---------------------------------------------------------------------------------------------------------------------\

   ____    __________________________________________________________________________________
  //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
  \\__//¯¯¯                                                                              ¯¯¯\\
  //¯¯¯                                                             Michael C. Cook Sr.     //
  \\                                                                  Security Engineer  ___//¯¯\\
   ¯¯¯\\__________________________________________________________Secure Digits Plus LLC//¯¯\\__//
       ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
```