```
  ____    ____________________________________________________________________________________________________
 //--\\__//----------------------------------------------------------------------------------------------\\___
 \\__//--- [FightingEntropy(π)][2022.12.0]: 2022-12-15 11:10:48                                          ___//--\\
  ---\\____________________________________________________________________________________________//--\\__//
      ----------------------------------------------------------------------------------------------  ----


\____________________________________________________________________________________________________________/
  About /------------------------------------------------------------------------------------------------------\
/-------
```

https://github.com/mcc85s/FightingEntropy/blob/main/Version/2022.12.0/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

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 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.

[FightingEntropy(π)][2022.12.0]		
Version	Date	Guid
2022.10.1	12/14/2022 14:26:18	5e6c9634-1c88-49a2-8794-2970095d8793

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)

    # // ------------------------------------------------------------------
    # // | 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
{
    [Object]   $Caption
    [Object]  $Platform
    [Object] $PSVersion
    [Object]      $Type
    [Object]    $Output
    OS()
    {
        $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>"
    }
}

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

Class File
{
    Hidden [UInt32]    $Index
    [String]            $Type
    [String]            $Name
    [String]            $Hash
    [UInt32]          $Exists
    Hidden [String] $Fullname
    Hidden [String]   $Source
    Hidden [UInt32]    $Match
    Hidden [Object]  $Content
    File([UInt32]$Index,[String]$Type,[String]$Parent,[String]$Name,[String]$Hash)
    {
        $This.Index    = $Index
        $This.Type     = $Type
        $This.Name     = $Name
        $This.Fullname = "$Parent\$Name"
        $This.Hash     = $Hash
        $This.TestPath()
    }
    [String] FolderName()
    {
        Return @{

            Control  = "Control"
            Function = "Functions"
            Graphic  = "Graphics"

        }[$This.Type]
    }
    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] Delete()
    {
        $This.TestPath()

        If ($This.Exists)
        {
            [System.IO.File]::Delete($This.Fullname)
            $This.Exists = 0
        }
    }
    SetSource([String]$Source)
    {
        $Version = [Regex]::Matches($This.Fullname,"\d{4}\.\d{2}\.\d+").Value
        $This.Source   = "{0}/blob/main/Version/{1}/{2}/{3}?raw=true" -f $Source,
                                                                        $Version,
                                                                        $This.FolderName(),
```

```
                                                                    $This.Name
    }
    Download()
    {
        Try
        {
            $This.Content = Invoke-WebRequest $This.Source -UseBasicParsing | % Content
        }
        Catch
        {
            Throw "Exception [!] An unspecified error occurred"
        }
    }
    Write()
    {
        If (!$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[]]$This.Content)
            }
            Else
            {
                [System.IO.File]::WriteAllText($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)
            }
            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.File>"
    }
}
```

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

Class Folder
{
    Hidden [UInt32]    $Index
    [String]            $Type
    [String]            $Name
    [String]        $Fullname
    [UInt32]          $Exists
    Hidden [Object]    $Item
    Folder([UInt32]$Index,[String]$Type,[String]$Parent,[String]$Name)
    {
        $This.Index     = $Index
        $This.Type      = $Type
        $This.Name      = $Name
        $This.Fullname  = "$Parent\$Name"
        $This.Item      = @( )
        $This.TestPath()
    }
    Add([String]$Name,[Object]$Hash)
    {
        $File           = [File]::New($This.Item.Count,$This.Type,$This.Fullname,$Name,$Hash)
        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     += $File
    }
    TestPath()
    {
        If (!$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] Delete()
    {
        $This.TestPath()

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

```
    }
}

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

Class Manifest
{
    [String]       $Source
    [String]     $Resource
    Hidden [UInt32] $Depth
    Hidden [UInt32] $Total
    [Object]       $Output
    Manifest([String]$Source,[String]$Resource)
    {
        $This.Source   = $Source
        $This.Resource = $Resource
        $This.Output   = @( )

        # // ___________
        # // | Control |
        # // -----------

        $This.AddFolder("Control","Control")

        ("Computer.png"                    , "87EAB4F74B38494A960BEBF69E472AB0764C3C7E782A3F74111F993EA31D1075") ,
        ("DefaultApps.xml"                 , "939CE697246AAC96C6F6A4A285C8EE285D7C5090523DB77831FF76D5D4A31539") ,
        ("failure.png"                     , "59D479A0277CFFDD57AD8B9733912EE1F3095404D65AB630F4638FA1F40D4E99") ,
        ("FEClientMod.xml"                 , "B3EB870C6B4206D11C921E70C6D058777A5F69FD1D9DEA8B6071759CAFCD2593") ,
        ("FEServerMod.xml"                 , "55A881BFE436EF18C104BFA51ECF6D12583076D576BA3276F53A682E056ACA5C") ,
        ("header-image.png"                , "38F1E2D061218D31555F35C729197A32C9190999EF548BF98A2E2C2217BBCB88") ,
        ("MDTClientMod.xml"                , "C22C53DAAB87AAC06DC3AC64F66C8F6DF4B7EAE259EC5D80D60E51AF82055231") ,
        ("MDTServerMod.xml"                , "3724FE189D8D2CFBA17BC2A576469735B1DAAA18A83D1115169EFF0AF5D42A2F") ,
        ("MDT_LanguageUI.xml"              , "100B5CA10BCF99E2A8680C394266042DEA5ECA300FBDA33289F6E4A17E44CBCF") ,
        ("PSDClientMod.xml"                , "4175C9569C8DFC1F14BADF70395D883BDD983948C2A6633CBBB6611430A872C7") ,
        ("PSDServerMod.xml"                , "4175C9569C8DFC1F14BADF70395D883BDD983948C2A6633CBBB6611430A872C7") ,
        ("success.png"                     , "46757AB0E2D3FFFFDBA93558A34AC8E36F972B6F33D00C4ADFB912AE1F6D6CE2") ,
        ("vendorlist.txt"                  , "9BD91057A1870DB087765914EAA5057D673CDC33145D804BBF4B024A11D66934") ,
        ("Wifi.cs"                         , "698AA48C98F500C6ED98305BCCA3C59C52784A664E01526D965A07AB24E47A2A") ,
        ("zipcode.txt"                     , "45D5F4B9B50782CEC4767A7660583C68A6643C02FC7CC4F0AE5A79CCABE83021")| % {

            $This.Add(0,$_[0],$_[1])
        }

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

        $This.AddFolder("Function","Functions")

        ("Copy-FileStream.ps1"             , "51D78BCE84D5EC2FABF85F87078D8E42179B19195E546371FC439E4B6171A0B9") ,
        ("Get-AssemblyList.ps1"            , "F2D1C0AD58A91CBF432A2AC793C8CD1313EB6F1A61C50D681130322C358CDAE7") ,
        ("Get-ControlExtension.ps1"        , "BF83DAAF1D8D53A39A5C5402A6BA9DEEA3DF32D37F38214DD93D1EBBE3149420") ,
        ("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"               , "492CBAB21ACCB448864B382B140ED72514DEEB015C7BBD282C649B0CEE262DE7") ,
        ("Get-FEDCPromo.ps1"               , "0CE646644BD4DF76B40A98D1D01E297012CAE7AAFA5FEF92C36AD9AA4CD43D2A") ,
        ("Get-FEImageManifest.ps1"         , "03AD403FA17EE0702A8D8911F8B4BD7AABE5C6971363AF2FFADE6FF83918D57F") ,
        ("Get-FEModule.ps1"                , "EEC6136D19426728E09571D55B983667AC99139EC02229B539047B4894E7AFF4") ,
        ("Get-FENetwork.ps1"               , "552CC93F8F21BCC2CC3CB1F0EBD447690E7EC41B1D4A8372C0839997CE48906E") ,
        ("Get-FERole.ps1"                  , "0016BDDB9B0BA9BB59652440FE0B758D88BF42A887F93B275F57016CCE4999C8") ,
        ("Get-FESystemDetails.ps1"         , "FE2C64E64DD76EC5B1B26E7BF4CF64C861416383974E63FDFE4499B09206C97F") ,
        ("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-ViperBomb.ps1"               , "A4548F9C2C730AE0714528642CCCDF0D6F35C9DD8D75DEC357ED7F164B76357E") ,
            ("Get-WhoisUtility.ps1"            , "A677D8026F18FBFF78C614CD3FC71BD6BE46EDC142D66CF9402EABAB9D988DCE") ,
            ("Install-BossMode.ps1"            , "25524DA6A44325BBFC5B4D4A863DE607B417CD4F3F57666627ECD9CB295AA07A") ,
            ("Install-IISServer.ps1"           , "48F53BF8A3ECD087E7F395AA19F86D32849CD4F14B599F2AC6F7330F083E0D6C") ,
            ("Install-PSD.ps1"                 , "0E0513C6BA4D98D1786E8606ABD5F6198947ADD43757E14D8138650DAB8D367F") ,
            ("Invoke-cimdb.ps1"                , "08ADF0105650B39381145D9018FF451FF03451F46B7FA28F0DE17208BE12E6EB") ,
            ("New-EnvironmentKey.ps1"          , "B2F51FA6AFCFD499DE96CFD7458E216832B36204BB542FDB416471058603D04C") ,
            ("New-FEFormat.ps1"                , "C4BFF5D8FBAC5ADBB79FEF848CE64A3C333C351EC1F50AC02468FCC0341AAAF4") ,
            ("New-FEInfrastructure.ps1"        , "04C48E828FEF3DDCC6B07D914D088AB471B6C768C10F2DD38FD230A5B0566F67") ,
            ("Search-WirelessNetwork.ps1"      , "614FFE3CDC091001E46CEEBAF69AC2FE8C22D517E9F97DD85CBA8B037EC890AA") ,
            ("Set-ScreenResolution.ps1"        , "60EE87AE8A1ADE31C2530BF3EC8E4BC03221692E599750265CE807648F9583E9") ,
            ("Show-ToastNotification.ps1"      , "661B9C815FF1BAEEE4400F65126741177D6F5D122161EF0093309A9067B8344E") ,
            ("Update-PowerShell.ps1"           , "EEA4AEEC98B7049F7273CDDCF58FCBD1702DDDEE1EB3C11B71DD76D30879F662") ,
            ("Write-Theme.ps1"                 , "DF937F03130E85B90CC22301DBDF89718F0EB795104EC9DF443090FDDA3E8FA")| % {

                $This.Add(1,$_[0],$_[1])
            }

            # // ____________
            # // | Graphics |
            # // ------------

            $This.AddFolder("Graphic","Graphics")

            ("background.jpg"                  , "94FD6CB32F8FF9DD360B4F98CEAA046B9AFCD717DA532AFEF2E230C981DAFEB5") ,
            ("banner.png"                      , "057AF2EC2B9EC35399D3475AE42505CDBCE314B9945EF7C7BCB91374A8116F37") ,
            ("icon.ico"                        , "594DAAFF448F5306B8B46B8DB1B420C1EE53FFD55EC65D17E2D361830659E58E") ,
            ("OEMbg.jpg"                       , "D4331207D471F799A520D5C7697E84421B0FA0F9B574737EF06FC95C92786A32") ,
            ("OEMlogo.bmp"                     , "98BF79CAE27E85C77222564A3113C52D1E75BD63283988718730 72F6B363D1A8") ,
            ("PSDBackground.bmp"               , "05ABBABDC9F67A95D5A4AF466149681C2F5E8ECD68F11433D32F4C0D04446F7E") ,
            ("sdplogo.png"                     , "87C2B016401CA3F8F8FAD5F629AFB3553C4762E14CD60792823D388F87E2B16C")| % {

                $This.Add(2,$_[0],$_[1])
            }

            $This.Total = ($This.Output | % Item).Count
            $This.Depth = ([String]$This.Total).Length
        }
        Add([UInt32]$Index,[String]$Name,[String]$Hash)
        {
            $This.Output[$Index] | % {

                $_.Add($Name,$Hash)
                $_.Item[-1].SetSource($This.Source)
            }
        }
        AddFolder([String]$Type,[String]$Name)
        {
            $This.Output += [Folder]::New($This.Output.Count,$Type,$This.Resource,$Name)
        }
        [String] Status([UInt32]$Rank)
        {
            Return "({0:d$($This.Depth)}/{1})" -f ($Rank+1), $This.Total
        }
        [String] Percent([UInt32]$Rank)
        {
            Return "{0:n2}" -f (($Rank/$This.Total) * 100)
        }
        Refresh()
        {
            $This.Output | % { $_.TestPath(); $_.Item | % TestPath }
        }
        Install()
        {
            $This.Refresh()

            $This.Output | ? Exists -eq 0 | % Create

            $List = $This.Output | % Item
            ForEach ($X in 0..($List.Count-1))
```

```
            {
                $File = $List[$X]
                $File.TestPath()
                If (!$File.Exists)
                {
                    $File.Create()
                    $File.Download()
                    $File.Write()
                    $File.TestPath()
                }
                Write-Host ("Installed [~] {0} {1}% -> {2}" -f $This.Status($X),
                                                               $This.Percent($X),
                                                               $File.Name)
            }
        }
        Remove()
        {
            $This.Refresh()

            $List  = $This.Output | % Item
            ForEach ($X in 0..($List.Count-1))
            {
                $File = $List[$X]
                $File.TestPath()
                If ($File.Exists)
                {
                    $File.Delete()
                    $File.TestPath()
                }
                Write-Host ("Removed [+] {0} {1:n2}% -> {2}" -f $This.Status($X),
                                                                $This.Percent($X),
                                                                $File.Name)
            }

            $This.Output | ? Exists -eq 1 | % Delete
        }
        [Object] List()
        {
            Return @(ForEach ($Folder in $This.Output)
            {
                $Folder
                $Folder | % Item
            })
        }
        [Object] Files([UInt32]$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.Manifest>"
        }
    }

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

    Class Template
    {
        [String]      $Source
        [String]        $Name
        [String] $Description
        [String]      $Author
        [String]     $Company
        [String]   $Copyright
        [Guid]          $Guid
        [DateTime]      $Date
```

```
        [String]     $Caption
        [String]    $Platform
        [String]        $Type
        [String]    $Registry
        [String]    $Resource
        [String]      $Module
        [String]        $File
        [String]    $Manifest
        Template([Object]$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.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
    {
        [String] $Type
        [String] $Name
        [String] $Fullname
        [UInt32] $Exists
        Hidden [String] $Path
        RootProperty([String]$Name,[UInt32]$Type,[String]$Fullname)
        {
            $This.Type     = Switch ($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 (!$This.Exists)
            {
                Switch -Regex ($This.Name)
                {
                    "(Resource|Module)"
                    {
                        [System.IO.Directory]::CreateDirectory($This.Fullname)
                    }
                    "(File|Manifest)"
                    {
                        [System.IO.File]::Create($This.Fullname).Dispose()
                    }
                }

                $This.TestPath()
            }
        }
```

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

            If ($This.Exists)
            {
                Switch -Regex ($This.Name)
                {
                    "(Resource|Module)"
                    {
                        [System.IO.Directory]::Delete($This.Fullname)
                    }
                    "(File|Manifest)"
                    {
                        [System.IO.File]::Delete($This.Fullname)
                    }
                }
                $This.Exists = 0
            }
        }
        [String] ToString()
        {
            Return $This.Path
        }
    }

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

    Class Root
    {
        [Object] $Registry
        [Object] $Resource
        [Object]   $Module
        [Object]     $File
        [Object] $Manifest
        [Object] $Shortcut
        Root([String]$Version,[String]$Resource,[String]$Path)
        {
            $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]$Path)
        {
            Return [RootProperty]::New($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 |
```

```powershell
# // ----------------------------------------

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)
        Write-Host "Registry [+] Path: [$($This.Fullname)]"
    }
    Add([String]$Name,[Object]$Value)
    {
        If (!$This.Subkey)
        {
            Throw "Must create the subkey first."
        }

        $This.Subkey.SetValue($Name,$Value)
        Write-Host "Key [+] Property: [$Name], Value: [$Value]"
    }
    [Void] Delete()
    {
        If ($This.Key)
        {
            $This.Key.DeleteSubKeyTree($This.Name)
            Write-Host "Registry [-] Path [$($This.Fullname)"
        }
    }
    [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,[String]$Name,[Object]$Value)
    {
        $This.Index = $Index
        $This.Name  = $Name
        $This.Value = $Value
    }
    [String] ToString()
    {
        Return "<FightingEntropy.Module.RegistryKeyProperty>"
    }
}

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

Class RegistryKey
{
    [String] $Path
    [UInt32] $Exists
    [Object] $Property
    RegistryKey([Object]$Module)
    {
        $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              = @{ }
        $Object.PSObject.Properties | ? Name -notmatch ^PS | % {

            $Item          = $This.Key($Hash.Count,$_.Name,$_.Value)
            $Item.Exists   = $This.Exists
            $Hash.Add($Hash.Count,$Item)
        }

        Return $Hash[0..($Hash.Count-1)]
    }
    TestPath()
    {
        $This.Exists = Test-Path $This.Path
    }
    [String] Status([UInt32]$Rank)
    {
        $D = ([String]$This.Property.Count).Length
        Return "({0:d$D}/{1})" -f $Rank, $This.Property.Count
    }
    Install()
    {
        $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,[String]$Name,[Object]$Value)
    {
        Return [RegistryKeyProperty]::New($Index,$Name,$Value)
    }
    [Object] RegistryKeyTemp([String]$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
    {
        [String] $Type
        [String] $Name
        Hidden [String] $Fullname
        Hidden [String] $Source
        [String] $Hash
        [UInt32] $Match
        [String] $Compare
        ValidateFile([String]$Leaf,[Object]$File)
        {
            $This.Type     = $File.Type
            $This.Name     = $File.Name
            $This.Fullname = $File.Fullname
            $This.Hash     = $File.Hash
            $This.Source   = $File.Source

            # // ________________________
            # // | Temporary variables |
            # // ------------------------

            $Content       = Invoke-WebRequest $This.Source -UseBasicParsing | % Content
            $Target        = "{0}\{1}" -f $Env:Temp, $This.Name

            If ([System.IO.File]::Exists($Target))
            {
                [System.IO.File]::Delete($Target)
            }

            If ($This.Name -match "\.+(jpg|jpeg|png|bmp|ico)")
            {
                [System.IO.File]::WriteAllBytes($Target,[Byte[]]$Content)
            }
            Else
            {
                [System.IO.File]::WriteAllText($Target,$Content,[System.Text.UTF8Encoding]$False)
            }

            # // ________________________________________
```

```powershell
            # // | Get target hash and final comparison |
            # // ----------------------------------------

            $This.Compare  = $This.GetFileHash($Target)
            $This.Match    = $This.Hash -eq $This.Compare

            [System.IO.File]::Delete($Target)
        }
        [String] GetFileHash([String]$Path)
        {
            If (![System.IO.File]::Exists($Path))
            {
                Throw "Invalid path"
            }

            Return Get-FileHash $Path | % Hash
        }
    }

    # // ---------------------------------------------------
    # // | Container class for (manifest/file) validation |
    # // ---------------------------------------------------

    Class Validate
    {
        [Object] $Output
        Validate([Object]$Module)
        {
            $Hash     = @{ }
            ForEach ($Branch in $Module.Manifest.Output)
            {
                Write-Host ("Path [~] [{0}]" -f $Branch.Fullname)
                ForEach ($File in $Branch.Item)
                {
                    Write-Host "File [~] [$($File.Fullname)]"
                    $Hash.Add($Hash.Count,$This.ValidateFile($Branch.Name,$File))
                }
            }

            $This.Output = $Hash[0..($Hash.Count-1)]
        }
        [Object] ValidateFile([String]$Name,[Object]$File)
        {
            Return [ValidateFile]::New($Name,$File)
        }
        [String] BuildManifest()
        {
            $MaxName = ($This.Output.Name | Sort-Object Length)[-1]
            Return @( $This.Output | % {

                "            (`"{0}`"{1}, `"{2}`") ," -f $_.Name,
                (@(" ") * ($MaxName.Length - $_.Name.Length + 1) -join ''),
                $_.Hash

            }) -join "`n"
        }
    }

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

    Class Installer
    {
        [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) 2022 (mcc85s/mcc85sx/sdp). All rights reserved."
        [Guid]           $Guid = "5e6c9634-1c88-49a2-8794-2970095d8793"
        [DateTime]       $Date = "12/14/2022 14:26:18"
        [Version]     $Version = "2022.12.0"
```

```
        [Object]          $OS
        [Object]        $Root
        [Object]    $Manifest
        [Object]    $Registry
        Installer([UInt32]$Mode)
        {
            If ($Mode -eq 0)
            {
                $This.Write("Loading [~] $($This.Label())")
            }

            $This.OS       = $This.GetOS()

            If ($Mode -eq 0)
            {
                Write-Host "[+] Operating System"
            }

            $This.Root     = $This.GetRoot()
            If ($Mode -eq 0)
            {
                Write-Host "[+] Module Root"
            }

            $This.Manifest = $This.GetManifest($This.Source,$This.Root.Resource)
            If ($Mode -eq 0)
            {
                Write-Host "[+] Module Manifest"
            }

            $This.Registry = $This.GetRegistry()
            If ($Mode -eq 0)
            {
                Write-Host "[+] Module Registry"
            }
        }
        [Object] NewVersion([String]$Version)
        {
            If ($Version -notmatch "\d{4}\.\d{2}\.\d+")
            {
                Throw "Invalid version entry"
            }

            Return [FEVersion]::New($True,$Version)
        }
        [Object[]] Versions()
        {
            $Markdown = Invoke-RestMethod "$($This.Source)/blob/main/README.md?raw=true"
            Return $Markdown -Split "`n" | ? { $_ -match "^\|\s\*\*\d{4}\.\d{2}\.\d+\*\*" } | % { [FEVersion]$_ }
        }
        [String] Label()
        {
            Return "{0}[{1}]" -f $This.Name, $This.Version.ToString()
        }
        [Object] Template()
        {
            Return [Template]::New($This)
        }
        [Object] GetOS()
        {
            Return [OS]::New()
        }
        [Object] GetRoot()
        {
            $Resource = $Env:ProgramData,
                        $This.Company,
                        "FightingEntropy",
                        $This.Version.ToString() -join "\"
            $Path     = Switch -Regex ($This.OS.Type)
            {
                ^Win32_ { $Env:PSModulePath -Split ";" -match [Regex]::Escape($Env:Windir) }
                Default { $Env:PSModulePath -Split ":" -match "PowerShell"                 }
            }
```

```
        Return [Root]::New($This.Version,$Resource,$Path)
    }
    [Object] GetManifest([String]$Source,[String]$Resource)
    {
        Return [Manifest]::New($Source,$Resource)
    }
    [Object] GetRegistry()
    {
        Return [RegistryKey]::New($This)
    }
    [Object] GetFEVersion()
    {
        Return [FEVersion]::New("| $($This.Version) | $($This.Date) | $($This.Guid) |")
    }
    [Void] Write([String]$Message)
    {
        [ThemeStack]::New($Message)
    }
    [Void] Write([UInt32]$Slot,[String]$Message)
    {
        [ThemeStack]::New($Slot,$Message)
    }
    [Object] File([String]$Type,[String]$Name)
    {
        Return $This.Manifest.List() | ? Type -eq $Type | ? 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] Refresh()
    {
        # // ___________________________________________
        # // | Tests all manifest (folder/file) entries |
        # // -------------------------------------------

        $This.Manifest.Output | % { $_.TestPath(); $_.Item | % TestPath }

        $This.Registry.TestPath()
        If ($This.Registry.Exists)
        {
            $This.Root.Registry.Exists = 1
        }
        $This.Root.Manifest.TestPath()
        $This.Root.File.TestPath()
        $This.Root.Module.TestPath()
    }
    [Void] Remove()
    {
        $This.Write(1,"Removing [~] $($This.Label())")

        # // ___________________________________________
        # // | Removing [Module]: (Manifest/File/Path) |
        # // -------------------------------------------

        "Shortcut","Manifest","File","Module" | % {

            $Item = $This.Root.$_
            $Item.Remove()
            Write-Host "Removed [+] $_ | $($Item.Fullname)"
        }

        # // ___________________________________________
        # // | Removing [Manifest/Registry]: (Content/Path) |
```

```
            # // ------------------------------------------------

            "Manifest","Registry" | % {

                Write-Host "Removing [~] $_"
                $This.$_.Remove()
                Write-Host "Removed [+] $_"
            }

            $This.Write(1,"Removed [+] $($This.Label())")
        }
        [Void] Install()
        {
            $This.Write(2,"Installing [~] $($This.Label())")

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

            $This.Manifest.Install()
            $This.Registry.Install()
            $This.Root.Module.Create()
            $This.Root.File.Create()

            # // ----------------------
            # // | Build the PSM/PSD |
            # // ----------------------

            $This._Module()

            # // ------------------------------------------------
            # // | Installs a shortcut to the module console |
            # // ------------------------------------------------

            $Com  = New-Object -ComObject WScript.Shell
            $Item = $Com.CreateShortcut($This.Root.Shortcut.Path)

            # // ----------------------------------
            # // | Assigns details to the shortcut |
            # // ----------------------------------

            $Item.TargetPath   = "PowerShell"

            $Command           = 'Add-Type -AssemblyName PresentationFramework',
                                 'Import-Module FightingEntropy',
                                 '$Module = Get-FEModule',
                                 '$Module' -join ";"
            $Item.Arguments    = "-NoExit -ExecutionPolicy Bypass -Command $Command"

            $Item.Description  = $This.Description
            $Item.IconLocation = $This._Graphic("icon.ico").Fullname
            $Item.Save()

            # // ------------------------------------------------------
            # // | Assigns administrative privileges to the shortcut |
            # // ------------------------------------------------------

            $Bytes             = [System.IO.File]::ReadAllBytes($This.Root.Shortcut)
            $Bytes[0x15]       = $Bytes[0x15] -bor 0x20
                                 # Set [byte] (21/0x15) bit 6 (0x20) ON... or else.
            [System.IO.File]::WriteAllBytes($This.Root.Shortcut, $Bytes)

            $This.Root.Shortcut.TestPath()

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

            $This.Write(2,"Installed [+] $($This.Label())")
        }
        [Void] Update()
        {
            $This.Root.File.Remove()
            $This.Root.Manifest.Remove()

            ForEach ($File in $This.Manifest.Output | % Item)
```

```
        {
            $Hash = Get-FileHash $File.Fullname | % Hash
            If ($Hash -ne $File.Hash)
            {
                $Message = @(
                "Exception [!] Type: [{0}]" -f $File.Type;
                "               File: [{0}]" -f $File.Fullname;
                "               Hash: [{0}]" -f $File.Hash;
                "           Mismatch: [{0}]" -f $Hash)

                Switch ((Get-Host).UI.PromptForChoice($Message,"Replace...?",@("&Yes","&No"),1))
                {
                    0
                    {
                        $File.Hash = $Hash
                        Write-Host ("Updated [+] File: [{0}]" -f $File.Name)
                        $File.GetContent()
                    }

                    1
                    {
                        Throw ("Exception [!] Hash mismatch, file: [{0}]" -f $File.Name)
                    }
                }
            }
        }

        $This._Module()
    }
    [Void] _Module()
    {
        # // ___________________
        # // | PowerShell Full |
        # // -------------------

        If ($This.Root.Resource.Exists)
        {
            # // ______________________________________________
            # // | Write the module file to disk using PSM() |
            # // ----------------------------------------------

            [System.IO.File]::WriteAllLines($This.Root.File.Fullname,
                                            $This.PSM(),
                                            [System.Text.UTF8Encoding]$False)

            # // ____________________________________
            # // | Splat the Module Manifest params |
            # // ------------------------------------

            $Splat = $This.PSDParam()

            # // ________________________________________________
            # // | Write the PowerShell module manifest to disk |
            # // ------------------------------------------------

            New-ModuleManifest @Splat

            $This.Root.Manifest.TestPath()
        }

        # // ____________________________________________________________________
        # // | Todo | PS Core | PS Server | <- Just a manner of file selection |
        # // --------------------------------------------------------------------
    }
    [String] PSM()
    {
        $F  = @( )

        # // __________
        # // | 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>"
            $This.Manifest.Files(1)  | % {

                $F += "# <{0}/{1}>" -f $_.Type, $_.Name
                $F += "# {0}" -f $_.Fullname
                If (!$_.Content)
                {
                    $_.GetContent()
                }
                $F += $_.Content
                $F += "# </{0}/{1}>" -f $_.Type, $_.Name
            }
            $F += "# </Functions>"
            $F += "Write-Theme -InputObject `"Module [+] [FightingEntropy(`$([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] Validation()
        {
            $This.Write(3,"Validation [~] Module manifest")

            $Validate = [Validate]::New($This)
            $Ct       = $Validate.Output | ? Match -eq 0

            Switch ($Ct.Count)
            {
                {$_ -eq 0}
                {
                    $This.Write(3,"Validation [+] All files passed validation")
                }
                {$_ -gt 0}
                {
                    $This.Write(1,"Validation [!] ($($Ct.Count)) files failed validation")
                    $Ct
                }
            }

            Return $Validate
        }
```

```
        [String] ToString()
        {
            Return "<FightingEntropy.Module.Installer>"
        }
    }

    [Installer]::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) 2022 (mcc85s/mcc85sx/sdp). All rights reserved.    |
    | Guid        : 5e6c9634-1c88-49a2-8794-2970095d8793                   |
    | Date        : 12/14/2022 14:26:18                                    |
    | Version     : 2022.12.0                                              |
    | 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
    -------   ----                ----
    2022.12.0 12/14/2022 14:26:18 5e6c9634-1c88-49a2-8794-2970095d8793


                                                                                    _________/
\_______________________________________________________________________________________/ Example
  Signature /-----------------------------------------------------------------------------------\
/-----------

    ------------------------------------------------------------------------------
    | Michael C. Cook Sr. | Security Engineer | Secure Digits Plus LLC | 2022-12-15 11:10:48 |
    ------------------------------------------------------------------------------
                                                                                    __________/
\_____________________________________________________________________________________/ Signature
/-------------------------------------------------------------------------------------------\
#>
```