[Programmatic Arithmetic]: 09/18/2023

Introduction

[Objective]: My system hard drive on my laptop is nearly full. Use some of the classes I've written in other (functions/classes) to transfer files that are NOT already in my [storage backup drive], to the [storage backup drive].

Class [VideoFileSize]

```
# // ═══════════════════════════════════════════
# // | Converts a byte size to a descriptive object |
# // ═══════════════════════════════════════════

Class VideoFileSize
{
    [String]   $Type
    [UInt64]  $Bytes
    [String]   $Unit
    [String]   $Size
    VideoFileSize([String]$Type,[UInt64]$Bytes)
    {
        $This.Type   = $Type
        $This.Bytes  = $Bytes
        $This.GetUnit()
        $This.GetSize()
    }
    GetUnit()
    {
        $This.Unit   = Switch ($This.Bytes)
        {
            {$_ -lt 1KB}                {     "Byte" }
            {$_ -ge 1KB -and $_ -lt 1MB} { "Kilobyte" }
            {$_ -ge 1MB -and $_ -lt 1GB} { "Megabyte" }
            {$_ -ge 1GB -and $_ -lt 1TB} { "Gigabyte" }
            {$_ -ge 1TB}                { "Terabyte" }
        }
    }
    GetSize()
    {
        $This.Size   = Switch -Regex ($This.Unit)
        {
            ^Byte     {     "{0} B" -f  $This.Bytes      }
            ^Kilobyte { "{0:n2} KB" -f ($This.Bytes/1KB) }
            ^Megabyte { "{0:n2} MB" -f ($This.Bytes/1MB) }
            ^Gigabyte { "{0:n2} GB" -f ($This.Bytes/1GB) }
            ^Terabyte { "{0:n2} TB" -f ($This.Bytes/1TB) }
        }
    }
    [String] GetLabel()
    {
        $Label = Switch ($This.Unit)
        {
            Byte     { "1"   }
            Kilobyte { "1KB" }
            Megabyte { "1MB" }
            Gigabyte { "1GB" }
            Terabyte { "1TB" }
        }
        Return $Label
    }
    [String] ToString()
    {
        Return $This.Size
    }
}
```

Class [VideoFileSize]

```
  Class [VideoFileDate] /-----------------------------------------------------------------------------------------------\
/------------------------

    # // ============================================================
    # // | Converts a [DateTime] object to a consumable format |
    # // ============================================================

    Class VideoFileDate
    {
        [String] $Type
        Hidden [DateTime] $Real
        [String] $Date
        VideoFileDate([String]$Type,[DateTime]$Real)
        {
            $This.Type = $Type
            $This.Real = $Real
            $This.Date = $Real.ToString("MM/dd/yyyy HH:mm:ss")
        }
        [String] ToString()
        {
            Return $This.Date
        }
    }

                                                                                                    _____________________/
\__________________________________________________________________________________________________/ Class [VideoFileDate]
  Class [VideoFileProgress] /-----------------------------------------------------------------------------------------\
/--------------------------

    # // ================================================================
    # // | Converts an index and rounded input to progress index |
    # // ================================================================

    Class VideoFileProgress
    {
            [UInt32]  $Index
            [UInt32]   $Slot
            [String] $Status
            VideoFileProgress([UInt32]$Index,[Object]$Slot)
            {
                $This.Index  = $Index
                $This.Slot   = $Slot
                $This.Status = "{0:p}" -f ($Index/100)
            }
            [String] ToString()
            {
                Return $This.Status
            }
    }

                                                                                                    _____________________/
\__________________________________________________________________________________________________/ Class [VideoFileProgress]
  Class [VideoFileEntry] /-------------------------------------------------------------------------------------------\
/------------------------

    # // ================================================================
    # // | Represents a file system object with the above nested classes |
    # // ================================================================

    Class VideoFileEntry
    {
        [UInt32] $Index
        [String] $Source
        [Object] $Size
        Hidden [UInt32] $Folder
        Hidden [Object] $Create
        Hidden [Object] $Access
        Hidden [Object] $Write
        [String] $Name
        [String] $Fullname
        [String] $Hash
        VideoFileEntry([UInt32]$Index,[String]$Source,[Object]$Item)
        {
            $This.Index    = $Index
            $This.Source   = $Source
            $This.Size     = $This.VideoFileSize("File",$Item.Length)
            $This.Folder   = $Item.PSIsContainer
            $This.Create   = $This.VideoFileDate("Created",$Item.CreationTime)
            $This.Access   = $This.VideoFileDate("Accessed",$Item.LastAccessTime)
            $This.Write    = $This.VideoFileDate("Modified",$Item.LastWriteTime)
            $This.Name     = $Item.Name
            $This.Fullname = $Item.Fullname
        }
```

```
        [Object] VideoFileDate([String]$Type,[DateTime]$Real)
        {
            Return [VideoFileDate]::New($Type,$Real)
        }
        [Object] VideoFileSize([String]$Name,[UInt64]$Bytes)
        {
            Return [VideoFileSize]::New($Name,$Bytes)
        }
        [String] GetFileHash()
        {
            Return Get-FileHash $This.Fullname | % Hash
        }
        [Object] Full()
        {
            Return $This | Select-Object Index, Source, Folder, Create, Access, Write, Name, Fullname, Size, Hash
        }
        [String] ToString()
        {
            Return $This.Name
        }
    }

\_____________________________________________________________________________________________/ Class [VideoFileEntry]
  Class [VideoFileIndex] /-------------------------------------------------------------------------------------------\
/------------------------

    # // ============================================================
    # // | Represents a file system object list, as well as type and path |
    # // ============================================================

    Class VideoFileIndex
    {
        [String] $Type
        [String] $Path
        [Object] $Output
        VideoFileIndex([String]$Type,[String]$Path)
        {
            $This.Type   = $Type
            $This.Path   = $Path
            $This.Refresh()
        }
        Clear()
        {
            $This.Output = @( )
        }
        [Object] VideoFileEntry([UInt32]$Index,[String]$Source,[Object]$Item)
        {
            Return [VideoFileEntry]::New($Index,$Source,$Item)
        }
        [Object] VideoFileProgress([UInt32]$Index,[Object]$Slot)
        {
            Return [VideoFileProgress]::New($Index,$Slot)
        }
        Refresh()
        {
            $Hash     = @{ }
            $List     = Get-ChildItem $This.Path -Recurse | Sort-Object Fullname
            $Step     = $List.Count/100
            $Track    = 0..100 | ? {$_ % 5 -eq 0} | % { $This.VideoFileProgress($_,[Math]::Round($_*$Step)) }
            $Slot     = $Track[0]
            $Activity = "Processing <{0}> files [~] Total: ({1})" -f $This.Type, $List.Count

            Write-Progress -Activity $Activity -Status $Slot.Status -PercentComplete $Slot.Index

            ForEach ($X in 0..($List.Count-1))
            {
                $Hash.Add($X,$This.VideoFileEntry($X,$This.Type,$List[$X]))
                If ($X -in $Track.Slot)
                {
                    $Slot = $Track | ? Slot -eq $X
                    Write-Progress -Activity $Activity -Status $Slot.Status -PercentComplete $Slot.Index
                }
            }

            $This.Output = $Hash[0..($Hash.Count-1)]

            Write-Progress -Activity $Activity -Status Complete -Completed
        }
    }

\_____________________________________________________________________________/ Class [VideoFileIndex]
```

```
  Class [VideoFileTransferProgress] /-------------------------------------------------------------------------------\
/-----------------------------------

    # // ================================================================
    # // | Converts an index, floating point, and total size to a progress index |
    # // ================================================================

    Class VideoFileTransferProgress
    {
        [UInt32]  $Index
        [Object]   $Size
        [String] $Status
        VideoFileTransferProgress([UInt32]$Index,[Float]$Step,[Object]$Total)
        {
            $This.Index  = $Index

            If ($Index -eq 0)
            {
                $Bytes = 0

            }
            Else
            {
                $Bytes = ($Total.Bytes*$Index)/100
            }

            $This.Size   = $This.VideoFileSize($Bytes)
            $This.Status = "{0:p}" -f ($Index/100)
        }
        [Object] VideoFileSize([UInt64]$Bytes)
        {
            Return [VideoFileSize]::New($This.Index,$Bytes)
        }
        [String] ToString()
        {
            Return $This.Status
        }
    }

                                                                                 _____________________________/
\________________________________________________________________________________/ Class [VideoFileTransferProgress]
  Class [VideoFileTransferPercent] /-------------------------------------------------------------------------------\
/-----------------------------------

    # // ================================================================================
    # // | Tracks start time, and allows percentage input to adjust ETA of transfer completion |
    # // ================================================================================

    Class VideoFileTransferPercent
    {
        [DateTime]   $Start
        [DateTime]     $Now
        [DateTime]     $End
        [Float]    $Percent
        [TimeSpan] $Elapsed
        [TimeSpan]  $Remain
        [TimeSpan]   $Total
        VideoFileTransferPercent([String]$Start)
        {
            $This.Start = [DateTime]$Start
        }
        VideoFilePercent([DateTime]$Start)
        {
            $This.Start = $Start
        }
        [Object] GetPercent([Float]$Percent)
        {
            $This.Now     = [DateTime]::Now
            $This.Elapsed = [TimeSpan]($This.Now-$This.Start)
            $This.Percent = $Percent
            $This.Total   = [TimeSpan]::FromSeconds(($This.Elapsed.TotalSeconds/$This.Percent)*100)
            $This.Remain  = $This.Total - $This.Elapsed
            $This.End     = [DateTime]($This.Now + $This.Remain)

            Return $This
        }
    }



                                                                                 _____________________________/
\________________________________________________________________________________/ Class [VideoFileTransferPercent]
```

```
  Class [VideoFileTransferStream] /-------------------------------------------------------------------------------------\
/---------------------------------

    # // ============================================================
    # // | For copying larger files with a progress indicator |
    # // ============================================================

    Class VideoFileTransferStream
    {
        Hidden [UInt32] $Id
        [String]      $Name
        [Object]    $Source
        [Object]    $Target
        [Byte[]]    $Buffer
        [Long]       $Total
        [UInt32]     $Count
        VideoFileTransferStream([UInt32]$Id,[String]$Source,[String]$Target)
        {
            $This.Id     = $Id
            $This.Name   = $Source | Split-Path -Leaf
            $This.Source = [System.IO.File]::OpenRead($Source)
            $This.Target = [System.IO.File]::OpenWrite($Target)

            $Splat            = @{

                Activity        = "Copying File [{0}]" -f $This.Name
                ParentId        = $This.Id
                Id              = 1
                Status          = "$Source -> $Target"
                PercentComplete = 0
            }

            Write-Progress @Splat
            Try
            {
                $This.Buffer  = [Byte[]]::New(4096)
                $This.Total   = $This.Count = 0
                Do
                {
                    $This.Count = $This.Source.Read($This.Buffer,0,$This.Buffer.Length)
                    $This.Target.Write($This.Buffer,0,$This.Count)
                    $This.Total += $This.Count
                    If ($This.Total % 1MB -eq 0)
                    {
                        $Splat              = @{

                            Activity        = "Copying File [{0}]" -f $This.Name
                            ParentId        = $This.Id
                            Id              = 1
                            Status          = "$Source -> $Target"
                            PercentComplete = ([Long]($This.Total*100/$This.Source.Length))
                        }

                        Write-Progress @Splat
                    }
                }
                While ($This.Count -gt 0)
            }
            Finally
            {
                $This.Source.Dispose()
                $This.Target.Dispose()

                $Splat            = @{

                    Activity        = "Copying File [{0}]" -f $This.Name
                    ParentId        = $This.Id
                    Id              = 1
                    Status          = "$Source -> $Target"
                    PercentComplete = 100
                }

                Write-Progress @Splat
            }
        }
    }

                                                                                                        _____________________/
\_______________________________________________________________________________________________________/ Class [VideoFileTransferStream]
```

```
  Class [VideoFileTransfer] /-------------------------------------------------------------------------------------------\
/---------------------------

    # // ======================
    # // | Controller class |
    # // ======================

    Class VideoFileTransfer
    {
        [Object]    $Source
        [Object]    $Target
        Hidden [UInt32] $Id
        [Object]  $Progress
        VideoFileTransfer([String]$Source,[String]$Target)
        {
            If (!$This.TestPath($Source))
            {
                Throw "Invalid <source> path"
            }

            ElseIf (!$This.TestPath($Target))
            {
                Throw "Invalid <target> path"
            }

            $This.Source = $This.VideoFileIndex("Source",$Source)
            $This.Target = $This.VideoFileIndex("Target",$Target)
            $This.Id     = Get-Random -Max 2147483647
        }
        [UInt32] TestPath([String]$Path)
        {
            Return [System.IO.Directory]::Exists($Path)
        }
        [Object] VideoFileIndex([String]$Type,[String]$Path)
        {
            Return [VideoFileIndex]::New($Type,$Path)
        }
        [Object] VideoFileProgress([UInt32]$Index,[Object]$Slot)
        {
            Return [VideoFileProgress]::New($Index,$Slot)
        }
        [Object] VideoFileTransferPercent()
        {
            Return [VideoFileTransferPercent]::New([DateTime]::Now)
        }
        [Object] VideoFileTransferProgress([UInt32]$Index,[Float]$Step,[Object]$Total)
        {
            Return [VideoFileTransferProgress]::New($Index,$Step,$Total)
        }
        [Object] VideoFileTransferStream([String]$Source,[String]$Target)
        {
            Return [VideoFileTransferStream]::New($This.Id,$Source,$Target)
        }
        [String] GetActivity([UInt32]$Count,[String]$Size)
        {
            $Item = "Transferring <Source -> Target> files [~] Total: ({0}), Size: ({1}), ETA: ({2})"
            Return $Item -f $Count, $Size, $This.Progress.Remain
        }
        Transfer()
        {
            $List          = $This.Source.Output | ? Name -notin $This.Target.Output.Name
            $Bytes         = ($List.Size.Bytes -join "+" | Invoke-Expression)
            $Total         = $This.VideoFileSize("Transfer",$Bytes)
            $Step          = $Total.Bytes/100
            $Track         = 0..100 | % { $This.VideoFileTransferProgress($_,$Step,$Total) }
            $Slot          = $Track[0]

            $This.Progress = $This.VideoFileTransferPercent()

            $Transferred   = 0

            $Splat         = @{

                Activity        = $This.GetActivity($List.Count,$Total.Size)
                Id              = $This.Id
                Status          = $Slot.Status
                PercentComplete = $Slot.Index
            }

            Write-Progress @Splat

            ForEach ($X in 0..($List.Count-1))
```

```
            {
                $File = $List[$X]
                If (!$File.Folder)
                {
                    $xSource = $File.Fullname
                    $xTarget = $xSource.Replace($This.Source.Path,$This.Target.Path)

                    # Test parent path
                    $xParent = $xTarget | Split-Path -Parent
                    If (![System.IO.Directory]::Exists($xParent))
                    {
                        [System.IO.Directory]::CreateDirectory($xParent)
                    }

                    $This.VideoFileTransferStream($xSource,$xTarget)

                    $Transferred = $Transferred + $File.Size.Bytes
                }

                If ($Transferred -gt $Slot.Size.Bytes)
                {
                    $Slot = $Track | ? { $_.Size.Bytes -gt $Transferred } | Select-Object -First 1

                    If ($Slot.Index -gt 0)
                    {
                        $This.Progress.GetPercent($Slot.Index)
                    }

                    $Splat         = @{

                        Activity        = $This.GetActivity($List.Count,$Total.Size)
                        Id              = $This.Id
                        Status          = $Slot.Status
                        PercentComplete = $Slot.Index
                    }

                    Write-Progress @Splat -EA 0
                }
            }

            Write-Progress -Activity Complete -Id $This.Id -Status Complete -Completed
        }
        [Object] VideoFileSize([String]$Type,[UInt64]$Bytes)
        {
            Return [VideoFileSize]::New($Type,$Bytes)
        }
        [String] ToString()
        {
            Return "<VideoFileTransfer>"
        }
    }
```

Class [VideoFileTransfer]

Explanation

So, with all of the (classes) listed up above, I can explain what's going on. Here's a list of the (classes):

[+] [Class VideoFileSize]
[+] [Class VideoFileDate]
[+] [Class VideoFileProgress]
[+] [Class VideoFileEntry]
[+] [Class VideoFileIndex]
[+] [Class VideoFileTransferProgress]
[+] [Class VideoFileTransferPercent]
[+] [Class VideoFileTransferStream]
[+] [Class VideoFileTransfer]

Everything revolves around the [VideoFileTransfer controller class], and everything else is embedded in that [controller class].

The variables [$Source], [$Target] and [$Ctrl] are meant to instantiate the [controller class].

```
 _____________________________________________________
/-----------------------------------------------------\
\ $Source = "C:\users\mcadmin\Videos"                 \
/ $Target = "E:\Videos"                               /
\ $Ctrl   = [VideoFileTransfer]::New($Source,$Target) \
/_____________________________________________________/
-------------------------------------------------------
```

And with that, the instantiation will test whether or not those paths exist.

If not → *an error will be thrown.*

If so → then the properties [Source] and [Target] will be populated with a [VideoFileIndex] object, which is essentially a fancy [list] object.

With each property [Source] and [Target], calling the [VideoFileIndex] class will automatically and *recursively* search for all (files+directories), and each file found will return a [VideoFileEntry] object, which has various properties that extend the standard file system object.

As each iteration of a (file+directory) is indexed, it pulls the [size], as well as the [date] for (CreationTime), (LastAccessTime), and (LastWriteTime).

The [VideoFileEntry] object CAN actually return the [hash value] of that particular [file], but this process takes a while for [very large files], so it should only be used when making comparisons between files that may overwrite one another... which won't happen with the way that the [$Ctrl.Transfer()] method in the controller class is written.

That's because it is only looking for the *names* of files.

Making comparisons between files that might be multiple gigabytes is a process that should only be reserved when need be, otherwise it will tax the system whenever it is used.

At any rate, once all of the files are indexed in the [Source] and [Target] property lists, then using the [$Ctrl.Transfer()] method will initialize all of this information that looks for all of the files, and I'll examine the block of code that does all of this, below.

Examination / Explanation

[Section 1]

```
$List          = $This.Source.Output | ? Name -notin $This.Target.Output.Name
$Bytes         = ($List.Size.Bytes -join "+" | Invoke-Expression)
$Total         = $This.VideoFileSize("Transfer",$Bytes)
$Step          = $Total.Bytes/100
$Track         = 0..100 | % { $This.VideoFileTransferProgress($_,$Step,$Total) }
$Slot          = $Track[0]
```

So, this area is capturing various properties.
It starts with the [$List] variable, which looks for any file that is not in the (target/destination) path.

The [VideoFileIndex] object actually extracts the names of any given file, and flattens them all into a list. Every time it does this, it is able to get the fullname, date, and size of each file.

If files have the *same name*, that's where this method will have an issue as it is written, which is why the [VideoFileIndex] object has a method named [.GetFileHash()], though that isn't being used yet.

This [$List] variable becomes an array that has a count, and, I can cast [$Bytes] by adding all of the byte lengths for all of those files.

[$Bytes = ($List.Size.Bytes -join "+" | Invoke-Expression)] actually joins all of those integers as a [String] with a plus sign in between each number. Then, when the expression is invoked, it will return the full total byte size for all of those files, to the variable [$Bytes].

Casting [$Total = $This.VideoFileSize("Transfer",$Bytes)] returns an [Object] that knows how much all of those bytes will break down to. So, in this instance , let's say we're talking about (167.91 GB), and that will take a while over [USB 2.0].

Type	Bytes	Unit	Size
Transfer	180291989668	Gigabyte	167.91 GB

[$Step = $Total.Bytes/100] just divides the total number into a floating point number, and then that is used as a parameter in the next line [$Track = 0..100 | % { $This.VideoFileTransferProgress($_,$Step,$Total) }].

That creates (101) objects that can chart the progress of the entire operation.

I'll filter some of them out and show multiples of (5).

```
 _____________________________________________________
/-----------------------------------------------------\
\ 0..100 | ? { $_ % 5 -eq 0 } | % { $Track[$_] } \
/                                                     /
\ Index Size      Status                              \
/ _____ ____      ______                              /
\     0 0 B       0.00%                               \
/     5 8.40 GB   5.00%                               /
\    10 16.79 GB  10.00%                              \
/    15 25.19 GB  15.00%                              /
\    20 33.58 GB  20.00%                              \
/    25 41.98 GB  25.00%                              /
\    30 50.37 GB  30.00%                              \
/    35 58.77 GB  35.00%                              /
\    40 67.16 GB  40.00%                              \
/    45 75.56 GB  45.00%                              /
\    50 83.95 GB  50.00%                              \
/    55 92.35 GB  55.00%                              /
\    60 100.74 GB 60.00%                              \
/    65 109.14 GB 65.00%                              /
\    70 117.54 GB 70.00%                              \
/    75 125.93 GB 75.00%                              /
\    80 134.33 GB 80.00%                              \
/    85 142.72 GB 85.00%                              /
\    90 151.12 GB 90.00%                              \
/    95 159.51 GB 95.00%                              /
\   100 167.91 GB 100.00%                             \
/_____________________________________________________/
-------------------------------------------------------
```

[Section 2] ═══

```
$This.Progress = $This.VideoFileTransferPercent()

$Transferred   = 0

$Splat         = @{

    Activity        = $This.GetActivity($List.Count,$Total.Size)
    Id              = $This.Id
    Status          = $Slot.Status
    PercentComplete = $Slot.Index
}

Write-Progress @Splat
```

═══

And, this area is basically setting up [progress indication] for the function (Write-Progress).

If we call it in the console without handing it a [percentage]…

```
 _____________________________________________
/---------------------------------------------\
\ PS Prompt:\> $Ctrl.Progress                 \
/                                             /
\ Start   : 9/18/2023 3:35:57 PM              \
/ Now     : 1/1/0001 12:00:00 AM              /
\ End     : 1/1/0001 12:00:00 AM              \
/ Percent : 0                                 /
\ Elapsed : 00:00:00                          \
/ Remain  : 00:00:00                          /
\ Total   : 00:00:00                          \
/_____________________________________________/
-----------------------------------------------
```

If we call it in the console AFTER handing it a [percentage]…

```
 _____________________________________________
/---------------------------------------------\
\ PS Prompt:\> $Ctrl.Progress.GetPercent(10) \
/                                             /
\ Start   : 9/18/2023 3:35:57 PM              \
/ Now     : 9/18/2023 3:41:28 PM              /
\ End     : 9/18/2023 4:31:07 PM              \
/ Percent : 10                                /
\ Elapsed : 00:05:31.0980431                  \
/ Remain  : 00:49:39.8819569                  /
\ Total   : 00:55:10.9800000                  \
/_____________________________________________/
-----------------------------------------------
```

But- HOW is it able to determine what [percentage] to hand it… especially if you're dealing with a bunch of files that have a determined size…?

[Section 3] ═══

```
ForEach ($X in 0..($List.Count-1))
{
    $File = $List[$X]
    If (!$File.Folder)
    {
        $xSource = $File.Fullname
        $xTarget = $xSource.Replace($Ctrl.Source.Path,$Ctrl.Target.Path)

        # Test parent path
        $xParent = $xTarget | Split-Path -Parent
        If (![System.IO.Directory]::Exists($xParent))
        {
            [System.IO.Directory]::CreateDirectory($xParent)
        }

        Copy-FileStream -Source $xSource -Destination $xTarget
        $Transferred = $Transferred + $File.Size.Bytes
    }

    …
```

═══

So, in this section, we're opening up a loop based on the number of objects that were captured to the [$List] variable. It does not account for whether the list has (0), or (1) object in it, mainly because while that may change the necessary logic slightly, the [assumption] being made is that someone is going to want to transfer [multiple large files], rather than just (1).

Because if that is the case, then just use the (Copy-FileStream) function, or the standard (Copy-Item).

The function (Copy-FileStream) is a custom function that deals with [FileStream] system objects, and is a part of [FightingEntropy(π)]. But you could also use (robocopy/xcopy/New-Object -ComObject Shell.Application).

Regardless, the class [VideoFileTransferStream] is nearly identical to the class that is in that function.

The helpful addition over (Copy-Item) is [progress indication], via (Write-Progress).

(Copy-Item) does NOT provide [progress indication], and it will essentially lock up the session until the file has been moved.

Whereas, (Copy-FileStream/[VideoFileTransferStream]) does the same thing- but it provides the [progress] of the operation for a [single file]. The objective of this series of classes, is to provide [progress indication] for transferring [multiple large files].

As it cycles through every single file in the [$List] variable, it will use [Regular Expressions] to replace the [source] portion of the fullname/path with the [target] portion, and it will split that target path to get its' [parent path], that way [System.IO.Directory] can test the path for its existence.

If it does not exist → it [creates that directory].

(If/When) it does exist → it begins [copying the file] from one place to the other.

At the tail end of the operation, the variable [$Transferred] is increased by that files' length, whereby allowing the next section to determine where in the [progress index] the operation actually is, which allows the [$Ctrl.Progress] variable to update itself and provide an [Estimated Time of Arrival/Completion].

[Section 4] ═══

```
    …
    If ($Transferred -gt $Slot.Size.Bytes)
    {
        $Slot = $Track | ? { $_.Size.Bytes -gt $Transferred } | Select-Object -First 1

        If ($Slot.Index -gt 0)
        {
            $Ctrl.Progress.GetPercent($Slot.Index)
        }

        $Activity = "Transferring <Source --> Target> ({0} files/{1}) [~] ETA: ({2})" -f $List.Count,
        $Total.Size,
        $Ctrl.Progress.Remain

        Write-Progress -Activity $Activity -Status $Slot.Status -PercentComplete $Slot.Index
    }
}
```

And in this section, each run through the loop is going to check whether or not the [$Transferred] variable has a value that is HIGHER than the currently selected slot. The slot being, a spot in the [progress index].

If it IS, the selected slot boosts itself to the first object in the [$Track] where the size in bytes is still greater than the total number of bytes that have been transferred.

At this point, [$Ctrl.Progress.GetPercent($Slot.Index)] allows the percentage to be put into the progress indicator class, which changes the ETA and whatnot.

At this juncture, during this demonstration, the (Write-Progress) was showing [progress indication] for the function (Copy-FileStream) rather than showing the outer (Write-Progress) progress, as well as the inner (Write-Progress) progress.

That is simply because the (function) should be turned back into a class that is embedded in the [controller class], that way, it would certainly show the [progress] for each item.

Examination

Conclusion

Anyway, that's gonna do it for this particular [document].

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