```
 ____    __________________________________________________________________________________________________
//¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
\\__//¯¯¯ Initialize-FeAdInstance [~]                                                              ___//¯¯\\
 ¯¯¯\\__________________________________________________________________________________________//¯¯\\__//
     ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
```

```
\________________________________________________________________________________________________________________/
  Introduction /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

This particular function is specifically meant to manage and populate an instance of Active Directory.

I recently made a couple of videos that showcase some of the work I've been doing with another function named “New-VmController”, as that function is a result of building virtualized servers to test the function “Get-FEDCPromo”.

All (3) of these functions are being worked on...

```
____________________________________________________________________________________________________________
| Get-FEDCPromo - For promoting a server to an Active Directory Domain Controller via (4) modes              |
| https://github.com/mcc85s/FightingEntropy/blob/main/Version/2022.12.0/Functions/Get-FEDCPromo.ps1          |
|==========================================================================================================|
| New-VmController - For building a lab environment of preconfigured virtual machines (Not in the module yet) |
| https://github.com/mcc85s/FightingEntropy/blob/main/Scripts/New-VmController.ps1                           |
|==========================================================================================================|
| Initialize-FeAdInstance - For populating an Active Directory Domain Controller with OU's, Groups, and Users |
| https://github.com/mcc85s/FightingEntropy/blob/main/Version/2022.12.0/Functions/Initialize-FeAdInstance.ps1 |
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

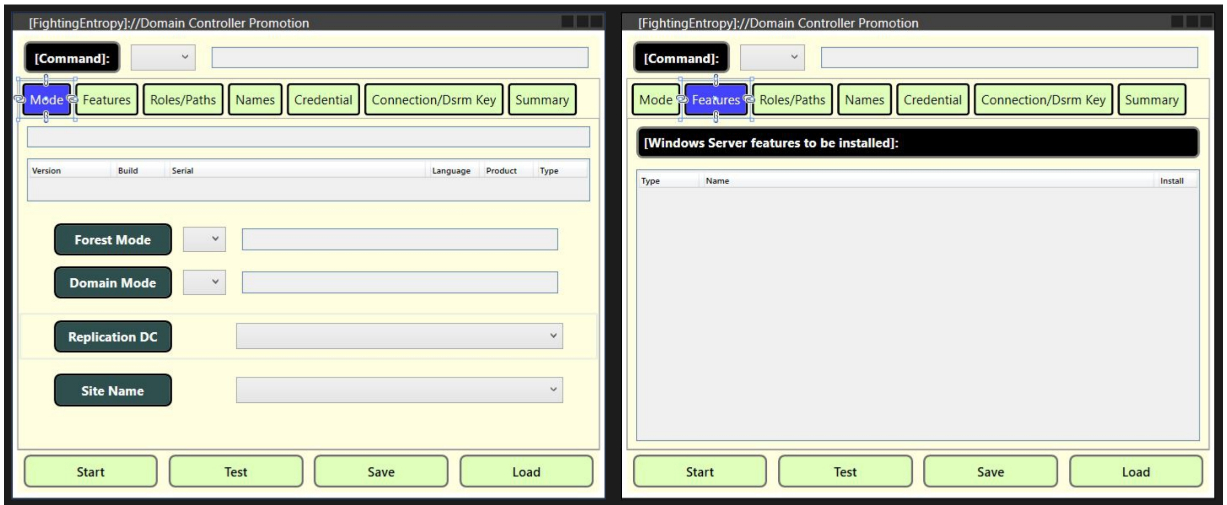
In this particular document, I'm only going to cover the last function, as I had to cobble it together in a day so that I could continue to develop the New-VmController script which I sorta covered in this video almost (2) weeks ago...

```
____________________________________________________________________________________________________
| 01/12/23 | 2023_0112-(PowerShell | Virtualization Lab + FEDCPromo) | https://youtu.be/9v7uJHF-cGQ |
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

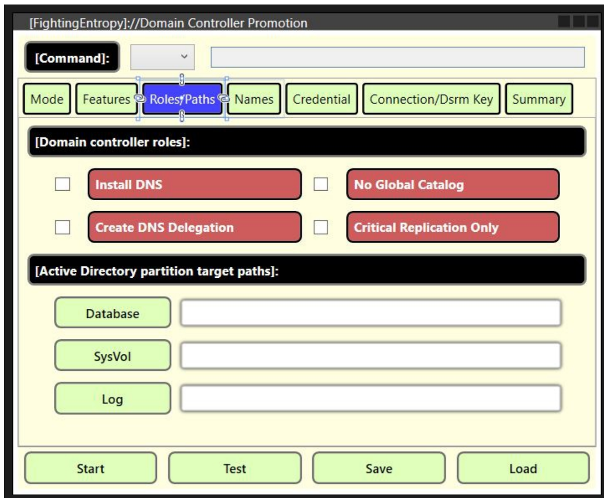
In that particular video, I was working with the GUI for “Get-FEDCPromo”.

I've since made many changes to Get-FEDCPromo, as well as officially named the script I was working on and it is called “New-VmController”.

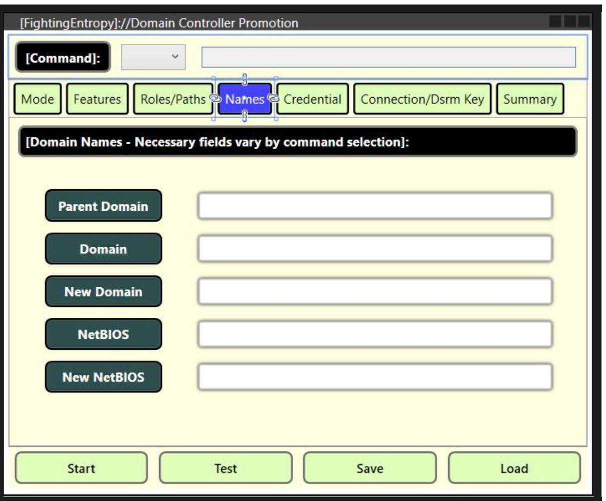
Here are some of the changes I've made to the function Get-FEDCPromo...



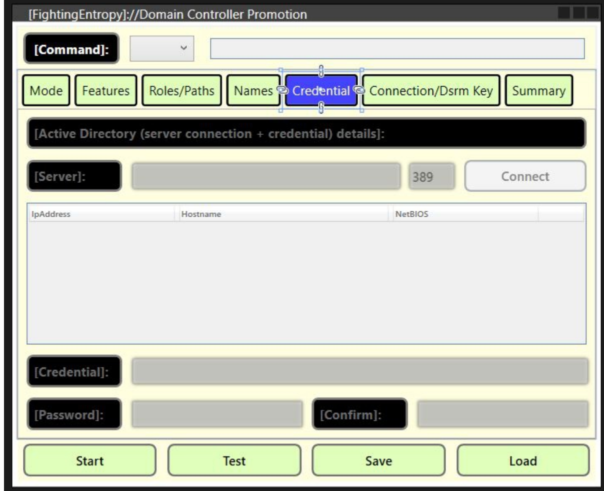
[Main] tab (same) [Features] tab (same)



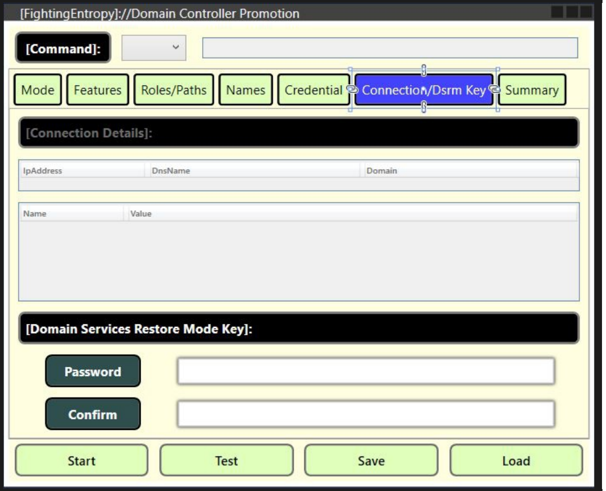
[Roles/Paths] tab (same)



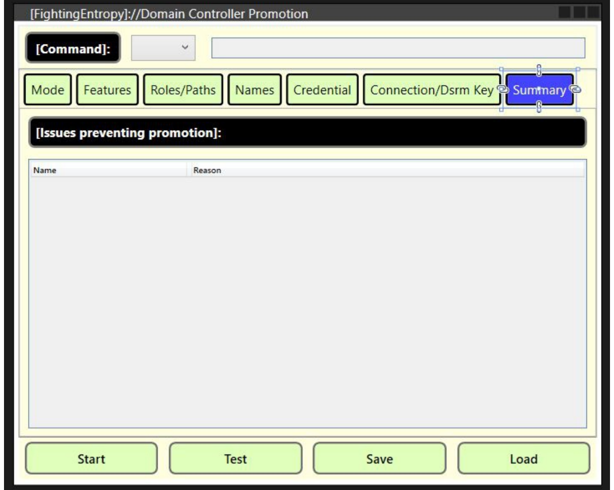
[Names] tab (same)



[Credential] tab (heavily altered)



[Connection/Dsrm key] Tab (new)



[Summary] tab (same)

In order to test the additions to Get-FEDCPromo, an available Active Directory domain must exist.

Before, the utility divided multiple aspects of the (scanning/login) process, and there are still a couple of additional things left to implement before that process is ready to (use/test).

```
At this juncture, I'm going to cover the function “Initialize-FeAdInstance”.
First, I will paste the function wrapper below without the embedded classes.
Then, I will cover each individual class.
```

```
<#
.SYNOPSIS
.DESCRIPTION
.LINK
.NOTES

 //==================================================================================================\\
//  Module     : [FightingEntropy()][2022.12.0]                                                       \\
\\  Date       : 2023-01-24 12:33:21                                                                  //
 \\==================================================================================================//

    FileName   : Initialize-FeAdInstance.ps1
    Solution   : [FightingEntropy()][2022.12.0]
    Purpose    : Populate Active Directory with authorized nodes, users, and computers
    Author     : Michael C. Cook Sr.
    Contact    : @mcc85s
    Primary    : @mcc85s
    Created    : 2023-01-23
    Modified   : 2023-01-24
    Demo       : N/A
    Version    : 0.0.0 - () - Finalized functional version 1
    TODO       : Finish and test

.Example
#>

Function Initialize-FeAdInstance
{
    Import-Module ActiveDirectory

   <# Classes go here #>

    [FeAdController]::New()
}
```

```
                                                                                                   ______________/
\__________________________________________________________________________________________________/ Introduction
  Enum [FeAdObjectSlotType] /------------------------------------------------------------------------------------\
/---------------------------
```

```
    # // ================================================================
    # // | Selected object types from Active Directory object list [Enum] |
    # // ================================================================

    Enum FeAdObjectSlotType
    {
        OrganizationalUnit
        Group
        User
    }
```

```
                                                                                    ___________________________/
\__________________________________________________________________________________/ Enum [FeAdObjectSlotType]
  Class [FeAdObjectSlotItem] /-----------------------------------------------------------------------------------\
/----------------------------
```

```
    # // ================================================================
    # // | Selected object types from Active Directory object list [Item] |
    # // ================================================================

    Class FeAdObjectSlotItem
    {
        [Uint32] $Index
        [String] $Type
        [String] $Description
        FeAdObjectSlotItem([String]$Type)
        {
            $This.Index = [UInt32][FeAdObjectSlotType]::$Type
            $This.Type  = $Type
        }
        [String] ToString()
        {
            Return "<FEModule.FeAdObjectSlotItem>"
        }
    }
```

```
                                                                                            ____________________________/
\__________________________________________________________________________________________/ Class [FeAdObjectSlotItem]
 Class [FeAdObjectSlotList] /---------------------------------------------------------------------------------------------\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

```
    # // ==================================================================
    # // | Selected object types from Active Directory object list [List] |
    # // ==================================================================

    Class FeAdObjectSlotList
    {
        [String]   $Name
        [UInt32]  $Count
        [Object] $Output
        FeAdObjectSlotList()
        {
            $This.Name = "FeAdObjectSlotList"
            $This.Refresh()
        }
        Clear()
        {
            $This.Output = @( )
            $This.Count  = 0
        }
        [Object] FeAdObjectSlotItem([String]$Type)
        {
            Return [FeAdObjectSlotItem]::New($Type)
        }
        Add([String]$Type)
        {
            $Item             = $This.FeAdObjectSlotItem($Type)
            $Item.Description = Switch ($Type)
            {
                OrganizationalUnit { "Base Active Directory container object"         }
                Group              { "Subordinate Active Directory collection object" }
                User               { "Subordinate Active Directory user object"       }
            }

            $This.Output     += $Item
            $This.Count       = $This.Output.Count
        }
        Refresh()
        {
            $This.Clear()

            ForEach ($Type in [System.Enum]::GetNames([FeAdObjectSlotType]))
            {
                $This.Add($Type)
            }
        }
        [String] ToString()
        {
            Return "({0}) <FEModule.FeAdObjectSlotList>" -f $This.Count
        }
    }
```

```
                                                                                             ____________________________/
\___________________________________________________________________________________________/ Class [FeAdObjectSlotList]
  Class [FeAdObjectItem] /------------------------------------------------------------------------------------------\
/------------------------
```

```
    # // =============================================
    # // | Represents an Active Directory node [Item] |
    # // =============================================

    Class FeAdObjectItem
    {
        Hidden [Object]     $Object
        [String]              $Name
        [String]              $Type
        [String]            $Exists
        [String]              $Guid
        [String] $DistinguishedName
        FeAdObjectItem([Object]$Object)
        {
            $This.Object            = $Object
            $This.Name              = $Object.Name
            $This.Type              = $Object.ObjectClass
            $This.Exists            = !!$Object
            $This.Guid              = $Object.ObjectGuid
            $This.DistinguishedName = $Object.DistinguishedName
        }
        [String] ToString()
        {
            Return "<FEModule.FeAdObjectItem>"
        }
    }
```

```
                                                                                                 ________________________/
\_______________________________________________________________________________________________/ Class [FeAdObjectItem]
  Class [FeAdObjectList] /------------------------------------------------------------------------------------------\
/------------------------
```

```
    # // =============================================
    # // | Represents an Active Directory node [List] |
    # // =============================================

    Class FeAdObjectList
    {
        [String]   $Name
        [UInt32]  $Count
        [Object] $Output
        FeAdObjectList()
        {
            $This.Name   = "FeAdObjectList"
            $This.Refresh()
        }
        Clear()
        {
            $This.Output = @( )
            $This.Count  = 0
        }
        Refresh()
        {
            $This.Clear()
            ForEach ($Item in Get-AdObject -Filter *)
            {
                $This.Add($Item)
            }
        }
        [Object] FeAdObjectItem([Object]$Object)
        {
            Return [FeAdObjectItem]::New($Object)
        }
        Add([Object]$Object)
        {
            $This.Output += $This.FeAdObjectItem($Object)
            $This.Count   = $This.Output.Count
        }
        [String] ToString()
        {
            Return "({0}) <FEModule.FeAdObjectList>" -f $This.Output.Count
        }
    }
```

```
                                                                                                ________________________/
\______________________________________________________________________________________________/ Class [FeAdObjectList]
  Class [FeAdLocation] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
# // ====================================================
# // | Represents an Active Directory location template |
# // ====================================================

Class FeAdLocation
{
    [String] $DisplayName
    [String] $StreetAddress
    [String] $City
    [String] $State
    [String] $PostalCode
    [String] $Country
    FeAdLocation([String]$Address,[String]$City,[String]$State,[String]$Zip,[String]$Country)
    {
        $This.StreetAddress = $Address
        $This.City          = $City
        $This.State         = $State
        $This.PostalCode    = $Zip
        $This.Country       = $Country

        $This.DisplayName   = $This.ToDisplayName()
    }
    [String] ToDisplayName()
    {
        $Split              = $This.City -Split " "
        $Item               = Switch ($Split.Count)
        {
            {$_ -eq 0} { $Null }
            {$_ -eq 1} { $This.City.Substring(0,2) }
            {$_ -gt 1} { ($Split | % { $_[0] }) -join '' }
        }

        Return "{0}-{1}-{2}-{3}" -f $Item, $This.State, $This.Country, $This.PostalCode
    }
    [String] ToString()
    {
        Return "<FEModule.FeAdLocation>"
    }
}
```

```
                                                                                                  ______________________/
\________________________________________________________________________________________________/ Class [FeAdLocation]
  Class [FeAdOrganizationalUnit] /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
# // ======================================================
# // | Represents an Active Directory Organizational Unit |
# // ======================================================

Class FeAdOrganizationalUnit
{
    Hidden [Object]         $Ou
```

```
        Hidden [Object]   $Location
        [String]              $Name
        [String]       $DisplayName
        [String]       $Description
        [String]     $StreetAddress
        [String]              $City
        [String]             $State
        [String]        $PostalCode
        [String]           $Country
        [UInt32]            $Exists
        [String] $DistinguishedName
        FeAdOrganizationalUnit([String]$Name,[String]$Description)
        {
            $This.Name              = $Name
            $This.DisplayName       = $This.ToDisplayName()
            $This.Description       = $Description
            $This.Check()
        }
        FeAdOrganizationalUnit([String]$Name,[String]$Description,[Object]$Location)
        {
            $This.Name              = $Name
            $This.DisplayName       = $This.ToDisplayName()
            $This.Description       = $Description
            $This.Location          = $Location
            $This.StreetAddress     = $Location.StreetAddress
            $This.City              = $Location.City
            $This.State             = $Location.State
            $This.PostalCode        = $Location.PostalCode
            $This.Country           = $Location.Country
            $This.Check()
        }
        FeAdOrganizationalUnit([Switch]$Flags,[Object]$Ou)
        {
            $This.Ou                = $Ou
            $This.Name              = $Ou.Name
            $This.DisplayName       = $Ou.DisplayName
            $This.Description       = $Ou.Description
            $This.StreetAddress     = $Ou.StreetAddress
            $This.City              = $Ou.City
            $This.State             = $Ou.State
            $This.PostalCode        = $Ou.PostalCode
            $This.Country           = $Ou.Country
            $This.Check()
        }
        Check()
        {
            $This.Get() | Out-Null
        }
        [Object] Get()
        {
            $This.Ou      = Get-AdOrganizationalUnit -Filter * -Properties * -EA 0 | ? Name -eq $This.Name
            $This.Exists  = !!$This.Ou
            $This.DistinguishedName = $This.Ou.DistinguishedName
            Return $This.Ou
        }
        Create()
        {
            $This.Check()
            If ($This.Exists)
            {
                Throw "Exception [!] Organizational unit already exists"
            }

            $Splat              = @{

                Name           = $This.Name
                DisplayName    = $This.DisplayName
                Description    = $This.Description
                StreetAddress  = $This.StreetAddress
                City           = $This.City
                State          = $This.State
                PostalCode     = $This.PostalCode
                Country        = $This.Country
            }

            New-AdOrganizationalUnit @Splat -Verbose
            $This.Check()
        }
        Remove()
        {
            $This.Check()
            If (!$This.Exists)
            {
                Throw "Exception [!] Organizational unit does not exist"
            }

            Set-ADObject    -Identity $This.DistinguishedName -ProtectedFromAccidentalDeletion 0 -EA 0
            Remove-ADObject -Identity $This.DistinguishedName -Confirm:0 -Recursive -Verbose     -EA 0
            $This.Check()
        }
        [String] ToDisplayName()
        {
            Return "[FightingEntropy({0})] <{1}>" -f [Char]960, $This.Name
        }
        [String] ToString()
        {
            Return "<FEModule.FeAdOrganizationalUnit>"
        }
    }
```

```
                                                                                        ________________________________/
\______________________________________________________________________________________/ Class [FeAdOrganizationalUnit]
  Class [FeAdGroup] /---------------------------------------------------------------------------------------------------\
/-------------------
```

```
# // ========================================
# // | Represents an Active Directory group |
# // ========================================

Class FeAdGroup
{
    Hidden [Object]      $Group
    [String]              $Name
    [String]       $DisplayName
    [String]        $GroupScope
    [String]     $GroupCategory
    [String]       $Description
    [String]              $Path
    [UInt32]            $Exists
    [String] $DistinguishedName
    FeAdGroup([String]$Name,[String]$Category,[String]$Scope,[String]$Description,[String]$Path)
    {
        $This.Name          = $Name
        $This.GroupCategory = $Category
        $This.GroupScope    = $Scope
        $This.Description   = $Description
        $This.Path          = $Path

        $This.DisplayName   = $This.ToDisplayName()

        $This.Check()
    }
    FeAdGroup([Switch]$Flags,[Object]$Group)
    {
        $This.Group         = $Group
        $This.Name          = $Group.Name
        $This.GroupCategory = $Group.GroupCategory
        $This.GroupScope    = $Group.GroupScope
        $This.Description   = $Group.Description
        $Label              = "CN={0}," -f $This.Name
        $This.Path          = $Group.DistinguishedName -Replace $Label, ""
        $This.DisplayName    = $Group.DisplayName

        $This.Check()
    }
    Check()
    {
        $This.Get() | Out-Null
    }
    [Object] Get()
    {
        $This.Group    = Get-AdGroup -Filter * -Properties * -EA 0 | ? Name -eq $This.Name
        $This.Exists   = !!$This.Group
        $This.DistinguishedName = $This.Group.DistinguishedName
        Return $This.Group
    }
    Create()
    {
        $This.Check()
        If ($This.Exists)
        {
            Throw "Exception [!] Group already exists"
        }

        $Splat              = @{

            Name           = $This.Name
            DisplayName    = $This.DisplayName
            Description    = $This.Description
            GroupScope     = $This.GroupScope
            GroupCategory  = $This.GroupCategory
            Path           = $This.Path
        }

        New-AdGroup @Splat -Verbose
        $This.Check()
    }
    Remove()
    {
        $This.Check()
        If (!$This.Exists)
        {
            Throw "Exception [!] Group does not exist"
        }

        Set-ADObject    -Identity $This.DistinguishedName -ProtectedFromAccidentalDeletion 0 -EA 0
        Remove-ADObject -Identity $This.DistinguishedName -Confirm:0 -Verbose     -EA 0
        $This.Check()
    }
    [String] ToDisplayName()
    {
        Return "[FightingEntropy({0})] <{1}>" -f [Char]960, $This.Name
    }
    [String] ToString()
    {
        Return "<FEModule.FeAdGroup>"
    }
}
```

```
                                                                                                   ___________________/
\_________________________________________________________________________________________________/ Class [FeAdGroup]
  Class [FeAdUser] /‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾\
/‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
```

```
# // ========================================
# // | Represents an Active Directory user |
# // ========================================

Class FeAdUser
{
    Hidden [Object]           $User
    [String]                  $Name
    [String]           $DisplayName
    [String]             $GivenName
    [String]              $Initials
    [String]               $Surname
    [String]           $Description
    [String]                $Office
    [String]          $EmailAddress
    [String]              $HomePage
    [String]         $StreetAddress
    [String]                  $City
    [String]                 $State
    [String]            $PostalCode
    [String]               $Country
    [String]        $SamAccountName
    [String]     $UserPrincipalName
    [String]           $ProfilePath
    [String]            $ScriptPath
    [String]         $HomeDirectory
    [String]             $HomeDrive
    [String]             $HomePhone
    [String]           $OfficePhone
    [String]           $MobilePhone
    [String]                   $Fax
    [String]                 $Title
    [String]            $Department
    [String]               $Company
    [String]                  $Path
    [UInt32]               $Enabled
    [UInt32]                $Exists
    [String]     $DistinguishedName
    FeAdUser([String]$Given,[String]$Initials,[String]$Surname,[String]$Sam,[String]$Path)
    {
        $This.GivenName         = $Given
        $This.Initials          = $Initials
        $This.Surname           = $Surname
        $This.DisplayName       = Switch ([UInt32]!$Initials)
        {
            0 { "{0} {1}. {2}" -f $Given, $Initials, $Surname } 1 { "{0} {1}" -f $Given, $Surname }
        }

        $This.Name              = $This.DisplayName
        $This.SamAccountName    = $Sam
        $This.UserPrincipalName = "{0}@{1}" -f $Sam, $This.Domain()
        $This.Path              = $Path

        $This.Check()
    }
    FeAdUser([Switch]$Flags,[Object]$User)
    {
        $This.User              = $User
        $This.Name              = $User.Name
        $This.DisplayName       = $User.DisplayName
        $This.GivenName         = $User.GivenName
        $This.Initials          = $User.Initials
        $This.Surname           = $User.Surname
        $This.Description       = $User.Description
        $This.Office            = $User.Office
        $This.EmailAddress      = $User.EmailAddress
        $This.HomePage          = $User.HomePage
        $This.StreetAddress     = $User.StreetAddress
        $This.City              = $User.City
        $This.State             = $User.State
        $This.PostalCode        = $User.PostalCode
        $This.Country           = $User.Country
        $This.SamAccountName    = $User.SamAccountName
        $This.UserPrincipalName = $User.UserPrincipalName
        $This.ProfilePath       = $User.ProfilePath
        $This.ScriptPath        = $User.ScriptPath
        $This.HomeDirectory     = $User.HomeDirectory
        $This.HomeDrive         = $User.HomeDrive
        $This.HomePhone         = $User.HomePhone
```

```
            $This.OfficePhone       = $User.OfficePhone
            $This.MobilePhone       = $User.MobilePhone
            $This.Fax               = $User.Fax
            $This.Title             = $User.Title
            $This.Department        = $User.Department
            $This.Company           = $User.Company

            $Label                  = "CN={0}," -f $This.Name
            $This.Path              = $User.DistinguishedName -Replace $Label, ""

            $This.Check()
        }
        Check()
        {
            $This.Get() | Out-Null
        }
        [Object] Get()
        {
            $This.User              = Get-AdUser -Filter * -Properties * -EA 0 | ? Name -eq $This.Name
            If ($This.User)
            {
                $This.Enabled       = [UInt32]$This.User.Enabled
            }
            $This.Exists            = !!$This.User
            $This.DistinguishedName = $This.User.DistinguishedName
            Return $This.User
        }
        Create()
        {
            $This.Check()
            If ($This.Exists)
            {
                Throw "Exception [!] User already exists"
            }

            $Splat                  = @{

                Name                = $This.Name
                DisplayName         = $This.DisplayName
                GivenName           = $This.GivenName
                Initials            = $This.Initials
                Surname             = $This.Surname
                SamAccountName      = $This.SamAccountName
                UserPrincipalName   = $This.UserPrincipalName
                Path                = $This.Path
            }

            New-AdUser @Splat -Verbose
            $This.Check()
        }
        Remove()
        {
            $This.Check()
            If (!$This.Exists)
            {
                Throw "Exception [!] User does not exist"
            }

            Set-ADObject    -Identity $This.DistinguishedName -ProtectedFromAccidentalDeletion 0 -EA 0
            Remove-ADObject -Identity $This.DistinguishedName -Confirm:0 -Verbose     -EA 0
            $This.Check()
        }
        [String] ToDisplayName()
        {
            Return "[FightingEntropy({0})] <{1}>" -f [Char]960, $This.Name
        }
        SetGeneral([String]$Description,[String]$Office,[String]$Email,[String]$Homepage)
        {
            $This.Description       = $Description
            $This.Office            = $Office
            $This.EmailAddress      = $Email
            $This.HomePage          = $Homepage
```

```powershell
        $Splat                  = @{ }

        ForEach ($Name in "Description","Office","EmailAddress","HomePage")
        {
            If ($This.$Name)
            {
                $Splat.Add($Name,$This.$Name)
            }
        }

        Set-AdUser -Identity $This.DistinguishedName @Splat -Verbose -EA 0
    }
    SetLocation([Object]$Location)
    {
        $This.StreetAddress     = $Location.StreetAddress
        $This.City              = $Location.City
        $This.State             = $Location.State
        $This.PostalCode        = $Location.PostalCode
        $This.Country           = $Location.Country

        $Splat                  = @{ }

        ForEach ($Name in "StreetAddress","City","State","PostalCode","Country")
        {
            If ($This.$Name)
            {
                $Splat.Add($Name,$This.$Name)
            }
        }

        Set-AdUser -Identity $This.DistinguishedName @Splat -Verbose -EA 0
    }
    SetProfile([String]$Profile,[String]$Script,[String]$Dir,[String]$Drive)
    {
        $This.ProfilePath       = $Profile
        $This.ScriptPath        = $Script
        $This.HomeDirectory     = $Dir
        $This.HomeDrive         = $Drive

        $Splat                  = @{ }

        ForEach ($Name in "ProfilePath","ScriptPath","HomeDirectory","HomeDrive")
        {
            If ($This.$Name)
            {
                $Splat.Add($Name,$This.$Name)
            }
        }

        Set-AdUser -Identity $This.DistinguishedName @Splat -Verbose -EA 0
    }
    SetTelephone([String]$xHome,[String]$Office,[String]$Mobile,[String]$Fax)
    {
        $This.HomePhone         = $xHome
        $This.OfficePhone       = $Office
        $This.MobilePhone       = $Mobile
        $This.Fax               = $Fax

        $Splat                  = @{ }

        ForEach ($Name in "HomePhone","OfficePhone","MobilePhone","Fax")
        {
            If ($This.$Name)
            {
                $Splat.Add($Name,$This.$Name)
            }
        }

        Set-AdUser -Identity $This.DistinguishedName @Splat -Verbose -EA 0
    }
    SetOrganization([String]$Title,[String]$Department,[String]$Company)
    {
        $This.Title             = $Title
        $This.Department        = $Department
        $This.Company           = $Company

        $Splat                  = @{ }

        ForEach ($Name in "Title","Department","Company")
        {
            If ($This.$Name)
            {
                $Splat.Add($Name,$This.$Name)
            }
        }

        Set-AdUser -Identity $This.DistinguishedName @Splat -Verbose -EA 0
    }
    SetAccountPassword([SecureString]$Pass)
    {
        $This.Check()

        If ($Pass.GetType().Name -ne "SecureString")
        {
            Throw "Invalid password entry"
        }

        If (!$This.Enabled)
        {
            Set-AdAccountPassword -Identity $This.DistinguishedName -NewPassword $Pass -Verbose -EA 0
            Set-AdUser -Identity $This.DistinguishedName -Enabled 1 -Verbose -EA 0
        }

        $This.Check()
    }
    SetPrimaryGroup([Object]$Group)
    {
        $Sid     = Get-AdObject -Identity $Group.DistinguishedName -Properties * | % ObjectSid
        $GroupId = $Sid.Value.Split("-")[-1]

        Set-AdObject -Identity $This.DistinguishedName -Replace @{ primaryGroupId = $GroupId } -Verbose
    }
    [String] Domain()
    {
        Return [Environment]::GetEnvironmentVariable("UserDnsDomain").ToLower()
    }
    [String] ToString()
    {
        Return "<FEModule.FeAdUser>"
    }
}
```

```
                                                                                                      ___________________/
\____________________________________________________________________________________________________/ Class [FeAdUser]
  Class [FeAdController] /----------------------------------------------------------------------------------------------\
/------------------------
```

```
# // ==================================================================
# // | Controller for Active Directory object (navigation/population) |
# // ==================================================================

Class FeAdController
{
    [Object]    $Types
    [Object]   $Object
    [Object] $Location
    FeAdController()
    {
        $This.Types  = $This.FeAdObjectSlotList()
        $This.Object = $This.FeAdObjectList()
    }
```

```
        Refresh()
        {
            $This.Object.Refresh()
        }
        [Object] FeAdObjectSlotList()
        {
            Return [FeAdObjectSlotList]::New()
        }
        [Object] FeAdObjectList()
        {
            Return [FeAdObjectList]::New()
        }
        [Object] FeAdLocation([String]$Address,[String]$City,[String]$State,[String]$Zip,[String]$Country)
        {
            Return [FeAdLocation]::New($Address,$City,$State,$Zip,$Country)
        }
        [Object] FeAdOrganizationalUnit([String]$Name,[String]$Desc)
        {
            Return [FeAdOrganizationalUnit]::New($Name,$Desc,$This.Location)
        }
        [Object] FeAdOrganizationalUnit([Switch]$Flags,[Object]$Object)
        {
            Return [FeAdOrganizationalUnit]::New($Flags,$Object)
        }
        [Object] FeAdGroup([String]$Name,[String]$Category,[String]$Scope,[String]$Desc,[String]$Path)
        {
            Return [FeAdGroup]::New($Name,$Category,$Scope,$Desc,$Path)
        }
        [Object] FeAdGroup([Switch]$Flags,[Object]$Object)
        {
            Return [FeAdGroup]::New($Flags,$Object)
        }
        [Object] FeAdUser([String]$Given,[String]$Initials,[String]$Surname,[String]$Sam,[String]$Path)
        {
            Return [FeAdUser]::New($Given,$Initials,$Surname,$Sam,$Path)
        }
        [Object] FeAdUser([Switch]$Flags,[Object]$Object)
        {
            Return [FeAdUser]::New($Flags,$Object)
        }
        [Object[]] Get([UInt32]$Index)
        {
            # // Throws if the index is not within bounds
            If ($Index -gt $This.Types.Output.Count)
            {
                Throw $This.ErrorSupport()
            }

            # // Refreshes the Active Directory node tree
            $This.Refresh()

            # // Selects the specific type
            $Type = $This.Types.Output[$Index].Type

            # // Returns the total list of objects with that type
            Return $This.Object.Output | ? Type -eq $Type
        }
        [String] ErrorSupport()
        {
            Return "Exception [!] Only supporting: (0:OrganizationalUnit/1:Group/2:User)"
        }
        SetLocation([String]$Address,[String]$City,[String]$State,[String]$Zip,[String]$Country)
        {
            $This.Location = $This.FeAdLocation($Address,$City,$State,$Zip,$Country)
        }
        AddAdOrganizationalUnit([String]$Name,[String]$Desc)
        {
            # // (Checks for/throws if) it does exist
            $xObject = $This.Get(0) | ? Name -eq $Name
            If ($xObject)
            {
                Throw "Exception [!] Organizational unit already exists"
        }

        # // Instantiates the object
        $Ou     = $This.FeAdOrganizationalUnit($Name,$Desc)

        # // Creates the object
        $Ou.Create()
    }
    [Object] GetAdOrganizationalUnit([String]$Name)
    {
        # // (Checks for/throws if) it does not exist
        $xObject = $This.Get(0) | ? Name -eq $Name
        If (!$xObject)
        {
            Throw "Exception [!] Organizational unit does not exist"
        }

        # // Obtains the object by distinguished name
        $Ou     = Get-AdOrganizationalUnit -Identity $xObject.DistinguishedName -Properties *

        # // Returns an instantiated version of the object
        Return $This.FeAdOrganizationalUnit([Switch]$True,$Ou)
    }
    RemoveAdOrganizationalUnit([String]$Name)
    {
        # // (Checks/Returns) instantiated version of the object
        $Ou     = $This.GetAdOrganizationalUnit($Name)

        # // Removes if found
        If ($Ou)
        {
            $Ou.Remove()
        }
    }
    AddAdGroup([String]$Name,[String]$Category,[String]$Scope,[String]$Desc,[String]$Path)
    {
        # // (Checks for/throws if) it does exist
        $xObject = $This.Get(1) | ? Name -eq $Name
        If ($xObject)
        {
            Throw "Exception [!] Group already exists"
        }

        # // Instantiates the object
        $Group  = $This.FeAdGroup($Name,$Category,$Scope,$Desc,$Path)

        # // Creates the object
        $Group.Create()
    }
    [Object] GetAdGroup([String]$Name)
    {
        # // (Checks for/throws if) it does exist
        $xObject = $This.Get(1) | ? Name -eq $Name
        If (!$xObject)
        {
            Throw "Exception [!] Group does not exist"
        }

        # // Obtains the object by distinguished name
        $Group = Get-AdGroup -Identity $xObject.DistinguishedName -Properties *

        # // Returns an instantiated version of the object
        Return $This.FeAdGroup([Switch]$True,$Group)
    }
    RemoveAdGroup([String]$Name)
    {
        # // (Checks/Returns) instantiated version of the object
        $Group = $This.GetAdGroup($Name)
        If ($Group)
        {
            $Group.Remove()
        }
        }
        [Object[]] GetAdPrincipalGroupMembership([String]$Name)
        {
            $Group = $This.GetAdGroup($Name)

            Return Get-AdPrincipalGroupMembership -Identity $Group.DistinguishedName
        }
        AddAdPrincipalGroupMembership([String]$GroupName,[String[]]$Names)
        {
            $Group = $This.Get(1) | ? Name -eq $GroupName
            $List  = $This.GetAdPrincipalGroupMembership($GroupName)

            ForEach ($Name in $Names)
            {
                If ($Name -notin $List.Name)
                {
                    $Splat       = @{

                        Identity = $Group.DistinguishedName
                        MemberOf = $Name
                    }

                    Add-AdPrincipalGroupMembership @Splat -EA 0 -Verbose
                }
            }
        }
        AddAdUser([String]$Given,[String]$Initials,[String]$Surname,[String]$Sam,[String]$Path)
        {
            # // Set template object
            $Name = Switch ([UInt32]!$Initials)
            {
                0 { "{0} {1}. {2}" -f $Given, $Initials, $Surname }
                1 { "{0} {1}" -f $Given, $Surname }
            }

            # // (Checks for/throws if) it does exist
            $xObject = $This.Get(2) | ? Name -eq $Name
            If ($xObject)
            {
                Throw "Exception [!] User already exists"
            }

            # // Instantiates the object
            $User  = $This.FeAdUser($Given,$Initials,$Surname,$Sam,$Path)

            # // Creates the object
            $User.Create()
        }
        [Object] GetAdUser([String]$Given,[String]$Initials,[String]$Surname)
        {
            # // Set template object
            $Name = Switch ([UInt32]!$Initials)
            {
                0 { "{0} {1}. {2}" -f $Given, $Initials, $Surname }
                1 { "{0} {1}" -f $Given, $Surname }
            }

            # // (Checks for/throws if) it does exist
            $xObject = $This.Get(2) | ? Name -eq $Name
            If (!$xObject)
            {
                Throw "Exception [!] User does not exist"
            }

            # // Obtains the object by distinguished name
            $User = Get-AdUser -Identity $xObject.DistinguishedName -Properties *

            # // Returns an instantiated version of the object
            Return $This.FeAdUser([Switch]$True,$User)
        }
        RemoveAdUser([String]$Given,[String]$Initials,[String]$Surname)
        {
            # // (Checks/Returns) instantiated version of the object
            $User = $This.GetAdUser($Given,$Initials,$Surname)
            If ($User)
            {
                $User.Remove()
            }
        }
        [Object[]] GetAdGroupMember([Object]$Group)
        {
            Return Get-AdGroupMember -Identity $Group.DistinguishedName
        }
        AddAdGroupMember([Object]$Group,[Object]$User)
        {
            $List = $This.GetAdGroupMember($Group)

            If ($User.DistinguishedName -notin $List.DistinguishedName)
            {
                $Splat       = @{

                    Identity = $Group.DistinguishedName
                    Members  = $User.DistinguishedName
                }

                Add-AdGroupMember @Splat -Verbose -EA 0
            }
        }
    }
```

```
                                                                                                ________________________/
\_______________________________________________________________________________________________/ Class [FeAdController]
  Output /--------------------------------------------------------------------------------------------------------------\
/--------
```

So, in this section I'll go over the (input/output) of the classes up above.

The idea here, is to make a class factory that is able to convert all of the embedded functions regarding Active Directory objects such as OrganizationalUnit, Group, and User, into an EXTENSION of those classes.

But also, the main method of going about retrieving these objects all comes down to the base command, “Get-ADObject”, and then being able to refresh that tree of objects in the least amount of time.

I tried a few methods, and this particular method seems to be the most responsive, and I'll go over the output and explain what it (is/does).

```
PS Prompt:\> $Password = "<|3adGuysSuck!>" | ConvertTo-SecureString -AsPlainText -Force
PS Prompt:\> $Ctrl = Initialize-FeAdInstance
PS Prompt:\> $Ctrl

Types                              Object                         Location
-----                              ------                         --------
(3) <FEModule.FeAdObjectSlotList> (247) <FEModule.FeAdObjectList>

PS Prompt:\>
```

So already, we're getting into the nitty gritty, by setting a variable named $Password with your standard-issue “<|3adGuysSuck!>” in plain text, which is immediately converted from a [String] into a [SecureString].

Then, we're casting the variable $Ctrl with the function “Initialize-FeAdInstance”.

As you can see, there are (3) properties.
The first is “Types”, the second is “Object”, and the last is “Location”.
The first (2) are showing that they've got content, and the last one is showing blank.
Let's populate this thing with a location.

```
PS Prompt:\> $Ctrl.SetLocation("1718 US-9","Clifton Park","NY",12065,"US")
PS Prompt:\> $Ctrl

Types                              Object                         Location
-----                              ------                         --------
(3) <FEModule.FeAdObjectSlotList> (247) <FEModule.FeAdObjectList> <FEModule.FeAdLocation>
```

```
PS Prompt:\>
```

Alright, so now there's a location there. Let's go over the output of those properties.

```
PS Prompt:\> $Ctrl.Types

Name               Count Output
----               ----- ------
FeAdObjectSlotList     3 {<FEModule.FeAdObjectSlotItem>, <FEModule.FeAdObjectSlotItem>... }

PS Prompt:\> $Ctrl.Types.Output

Index Type               Description
----- ----               -----------
    0 OrganizationalUnit Base Active Directory container object
    1 Group              Subordinate Active Directory collection object
    2 User               Subordinate Active Directory user object

PS Prompt:\>
```

That is what comes up with the property “Types”.
This is strictly meant for filtering out the things that are currently able to be managed via this function.

At some point, this function will eventually have more of the types that Active Directory handles.
For now, we're gonna focus on the (3) things that I've opted to use, in order to distribute:
(1) OU    → meant to hold (1) group → meant to provide (1) user...with DomainLocal administrative privileges.

What that (1) user will be able to DO once they have those privileges, is anyone's guess.
They'll effectively have total control over the (domain/forest), but they'll still be able to be easily managed.

I won't show ALL of the output for the “Object” property, but I'll select some of the entries.

```
PS Prompt:\> $Ctrl.Object

Name           Count Output
----           ----- ------
FeAdObjectList   247 {<FEModule.FeAdObjectItem>, <FEModule.FeAdObjectItem>, <FEModule.FeAdObjectItem>...}

PS Prompt:\> $Ctrl.Object.Output[0..10] | Format-Table

Name                                    Type      Exists Guid                                 DistinguishedName
----                                    ----      ------ ----                                 -----------------
securedigitsplus                        domainDNS True   921b527f-b6e6-47ae-9430-66b159d260cb DC=securedigit...
Users                                   container True   cd0e2c1d-a503-435b-a908-a59f710064f4 CN=Users,DC=se...
Allowed RODC Password Replication Group group     True   82d01769-fbdc-4663-bcf6-6d6d240b15ed CN=Allowed ROD...
Denied RODC Password Replication Group  group     True   65e4a07d-ec6d-44c9-925b-a8642ebdcc37 CN=Denied RODC...
Read-only Domain Controllers            group     True   ac531e6f-492f-4682-a0f8-1d8c5a5f027a CN=Read-only...
Enterprise Read-only Domain Controllers group     True   65cbab30-bc06-4868-9449-b5ba4ebdf751 CN=Enterprise...
Cloneable Domain Controllers            group     True   6fb6fe7b-9d69-4511-aa3c-86b9b717a632 CN=Cloneable...
Protected Users                         group     True   38490412-b8f2-4404-8d24-abe79c7007b6 CN=Protected...
Key Admins                              group     True   541a3063-0b79-4073-93e4-b4377a751d68 CN=Key Admins...
Enterprise Key Admins                   group     True   3df92a9a-7f2c-4f87-ab94-e77eca1bac59 CN=Enterprise...
DnsAdmins                               group     True   da67c012-e645-4fad-8ef7-ab440ef02047 CN=DnsAdmins,C...

PS Prompt:\>
```

Alright, these results are truncated, just in case anyone wanted to know.

```
PS Prompt:\> $Ctrl.Location

DisplayName   : CP-NY-US-12065
StreetAddress : 1718 US-9
City          : Clifton Park
State         : NY
PostalCode    : 12065
Country       : US

PS Prompt:\>
```

That's the address that I've selected.
It's the old county courthouse in landmark square, across from the Snyders restaurant, in Clifton Park NY.
Basically, it's one of the oldest buildings in town.

I'd like to make an organizational unit named "DevOps" meant for "Developer(s)/Operator(s)"

```
    # Add Organizational Unit
    $Ctrl.AddAdOrganizationalUnit("DevOps","Developer(s)/Operator(s)")

    # Get Organizational Unit
    $Ou     = $Ctrl.GetAdOrganizationalUnit("DevOps")
```

```
PS Prompt:\> # Add Organizational Unit
>> $Ctrl.AddAdOrganizationalUnit("DevOps","Developer(s)/Operator(s)")
VERBOSE: Performing the operation "New" on target "OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> # Get Organizational Unit
>> $Ou = $Ctrl.GetAdOrganizationalUnit("DevOps")
PS Prompt:\> $Ou

Name              : DevOps
DisplayName       : [FightingEntropy(π)] <DevOps>
Description       : Developer(s)/Operator(s)
StreetAddress     : 1718 US-9
City              : Clifton Park
State             : NY
PostalCode        : 12065
Country           : US
Exists            : 1
DistinguishedName : OU=DevOps,DC=securedigitsplus,DC=com

PS Prompt:\>
```

Thats the actual-factual output of the commands up above. We're not gonna stop there, because like I said, we want to create (1) group and (1) user that could theoretically cause all sorts of chaos, cacaphony, and mayhem.

I'd like to create a "Security" group named "Engineering" that's only available to "Global".
But also, I'd like this group to have the description of my company name, "Secure Digits Plus LLC" and I'd like the group to be placed within the $Ou up above, under it's DistinguishedName.

```
    # Add Group
    $Ctrl.AddAdGroup("Engineering","Security","Global","Secure Digits Plus LLC",$Ou.DistinguishedName)

    # Get Group
    $Group  = $Ctrl.GetAdGroup("Engineering")
```

```
PS Prompt:\> # Add Group
>> $Ctrl.AddAdGroup("Engineering","Security","Global","Secure Digits Plus LLC",$Ou.DistinguishedName)
VERBOSE: Performing the operation "New" on target "CN=Engineering,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> # Get Group
>> $Group  = $Ctrl.GetAdGroup("Engineering")
PS Prompt:\> $Group

Name              : Engineering
DisplayName       : [FightingEntropy(π)] <Engineering>
GroupScope        : Global
GroupCategory     : Security
Description       : Secure Digits Plus LLC
Path              : OU=DevOps,DC=securedigitsplus,DC=com
Exists            : 1
DistinguishedName : CN=Engineering,OU=DevOps,DC=securedigitsplus,DC=com

PS Prompt:\>
```

And, there it is.
The next thing that needs to be done with it, is to set its' principal group membership.

We can do THAT with the following command...

```
    # Add-AdPrincipalGroupMembership
    $Ctrl.AddAdPrincipalGroupMembership($Group.Name,@("Administrators","Domain Admins"))
```

```
PS Prompt:\> # Add-AdPrincipalGroupMembership
>> $Ctrl.AddAdPrincipalGroupMembership($Group.Name,@("Administrators","Domain Admins"))
VERBOSE: Adds all the specified member(s) to the specified group(s).
VERBOSE: Adds all the specified member(s) to the specified group(s).
PS Prompt:\> $Ctrl.GetAdPrincipalGroupMembership($Group.Name)

distinguishedName : CN=Administrators,CN=Builtin,DC=securedigitsplus,DC=com
GroupCategory     : Security
GroupScope        : DomainLocal
name              : Administrators
objectClass       : group
objectGUID        : 5545f235-b0aa-4946-99d3-04d97fe7006d
SamAccountName    : Administrators
SID               : S-1-5-32-544

distinguishedName : CN=Domain Admins,CN=Users,DC=securedigitsplus,DC=com
GroupCategory     : Security
GroupScope        : Global
name              : Domain Admins
objectClass       : group
objectGUID        : afd3e081-2a9b-48c5-8093-46a0a43f0aad
SamAccountName    : Domain Admins
SID               : S-1-5-21-1462195080-326095750-1738641541-512

PS Prompt:\>
```

It's important to note here, that these are methods that are accessing the default commands that do the same thing. However, by including them all within this controller class...? They're easier to use since the control class is fabricating extensions of the default Active Directory (base classes/management objects).

Last but not least, the user.
Since the user object is quite complex when all of the details are necessary...
...the method to create user objects has been split here between a bunch of various methods.

The default function DOES accept a LOT more parameters and values, however, these are the ones I've selected for now, to establish the (1) user that has DomainLocal Administrative privileges and can remotely access the domain and provide management as well as add other users, groups, service accounts, domain controllers, etc.

It certainly is possible to use the default administrator account to perform all of this administration...
However, that is typically frowned upon when dealing with larger corporate/business networks with a lot of users on mission critical infrastructure. In other words, it's not smart to use that account if you can help it, and DISABLING it once the prerequisites are in place, is a very good idea.

```
    # Add User
    $Ctrl.AddAdUser("Michael","C","Cook","mcook85",$Ou.DistinguishedName)
    # Get User
    $User   = $Ctrl.GetAdUser("Michael","C","Cook")
```

```
PS Prompt:\> # Add User
>> $Ctrl.AddAdUser("Michael","C","Cook","mcook85",$Ou.DistinguishedName)
VERBOSE: Performing the operation "New" on target "CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> # Get User
>> $User   = $Ctrl.GetAdUser("Michael","C","Cook")
PS Prompt:\> $User

Name              : Michael C. Cook
DisplayName       : Michael C. Cook
GivenName         : Michael
Initials          : C
Surname           : Cook
Description       :
Office            :
EmailAddress      :
HomePage          :
```

```
StreetAddress     :
City              :
State             :
PostalCode        :
Country           :
SamAccountName    : mcook85
UserPrincipalName : mcook85@securedigitsplus.com
ProfilePath       :
ScriptPath        :
HomeDirectory     :
HomeDrive         :
HomePhone         :
OfficePhone       :
MobilePhone       :
Fax               :
Title             :
Department        :
Company           :
Path              : OU=DevOps,DC=securedigitsplus,DC=com
Enabled           : 0
Exists            : 1
DistinguishedName : CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com

PS Prompt:\>
```

Now, that's a rather empty looking object right there.
It isn't exactly the full weight of the actual AdUser object, either.

But- there are other methods that allow the user to be updated with other information.

```
    # Set [User.General (Description, Office, Email, Homepage)]
    $User.SetGeneral("Beginning the fight against ID theft and cybercrime",
                     "<Unspecified>",
                     "michael.c.cook.85@gmail.com",
                     "https://github.com/mcc85s/FightingEntropy")
```

```
PS Prompt:\> # Set [User.General (Description, Office, Email, Homepage)]
>> $User.SetGeneral("Beginning the fight against ID theft and cybercrime",
>>                  "<Unspecified>",
>>                  "michael.c.cook.85@gmail.com",
>>                  "https://github.com/mcc85s/FightingEntropy")
VERBOSE: Performing the operation "Set" on target "CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> $User

Name              : Michael C. Cook
DisplayName       : Michael C. Cook
GivenName         : Michael
Initials          : C
Surname           : Cook
Description       : Beginning the fight against ID theft and cybercrime
Office            : <Unspecified>
EmailAddress      : michael.c.cook.85@gmail.com
HomePage          : https://github.com/mcc85s/FightingEntropy
StreetAddress     :
City              :
State             :
PostalCode        :
Country           :
SamAccountName    : mcook85
UserPrincipalName : mcook85@securedigitsplus.com
ProfilePath       :
ScriptPath        :
HomeDirectory     :
HomeDrive         :
HomePhone         :
OfficePhone       :
MobilePhone       :
Fax               :
Title             :
Department        :
```

```
Company           :
Path              : OU=DevOps,DC=securedigitsplus,DC=com
Enabled           : 0
Exists            : 1
DistinguishedName : CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com

PS Prompt:\>
```

```
Looks like we're really gettin' somewhere with setting this guy up with his information, like his
-Description  (what he's all about)
-Office       (where you might be able to find him)
-EmailAddress (where you could email him if the primary domain mail isn't up...)
-HomePage     (probably where this guy puts all of his work...)

Looks like his description is "Beginning the fight against ID theft and cybercrime"...
...sounds pretty intense.
```

```
    # Set [User.Address (StreetAddress, City, State, PostalCode, Country)]
    $User.SetLocation($Ctrl.Location)
```

```
PS Prompt:\> # Set [User.Address (StreetAddress, City, State, PostalCode, Country)]
>> $User.SetLocation($Ctrl.Location)
VERBOSE: Performing the operation "Set" on target "CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> $User

Name              : Michael C. Cook
DisplayName       : Michael C. Cook
GivenName         : Michael
Initials          : C
Surname           : Cook
Description       : Beginning the fight against ID theft and cybercrime
Office            : <Unspecified>
EmailAddress      : michael.c.cook.85@gmail.com
HomePage          : https://github.com/mcc85s/FightingEntropy
StreetAddress     : 1718 US-9
City              : Clifton Park
State             : NY
PostalCode        : 12065
Country           : US
SamAccountName    : mcook85
UserPrincipalName : mcook85@securedigitsplus.com
ProfilePath       :
ScriptPath        :
HomeDirectory     :
HomeDrive         :
HomePhone         :
OfficePhone       :
MobilePhone       :
Fax               :
Title             :
Department        :
Company           :
Path              : OU=DevOps,DC=securedigitsplus,DC=com
Enabled           : 0
Exists            : 1
DistinguishedName : CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com

PS Prompt:\>
```

```
So now we're able to deposit the location into various aspects of the Active Directory instance, which is not
unlike using a <Template> or an <Instance> object, after it has been populated. We're using an alternate method
of achieving the same end result, AND, this user can actually be used as a (template/instance).
```

```
    # Set [User.Profile (ProfilePath, ScriptPath, HomeDirectory, HomeDrive)]
    $User.SetProfile("","","","")
    # Set [User.Telephone (HomePhone, OfficePhone, MobilePhone, Fax)]
    $User.SetTelephone("","518-406-8569","518-406-8569","")
    # Set [User.Organization (Title, Department, Company)]
    $User.SetOrganization("CEO/Security Engineer","Engineering","Secure Digits Plus LLC")
```

```
PS Prompt:\> # Set [User.Profile (ProfilePath, ScriptPath, HomeDirectory, HomeDrive)]
>> $User.SetProfile("","","","")
VERBOSE: Performing the operation "Set" on target "CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> # Set [User.Telephone (HomePhone, OfficePhone, MobilePhone, Fax)]
>> $User.SetTelephone("","518-406-8569","518-406-8569","")
VERBOSE: Performing the operation "Set" on target "CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> # Set [User.Organization (Title, Department, Company)]
>> $User.SetOrganization("CEO/Security Engineer","Engineering","Secure Digits Plus LLC")
VERBOSE: Performing the operation "Set" on target "CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> $User

Name              : Michael C. Cook
DisplayName       : Michael C. Cook
GivenName         : Michael
Initials          : C
Surname           : Cook
Description       : Beginning the fight against ID theft and cybercrime
Office            : <Unspecified>
EmailAddress      : michael.c.cook.85@gmail.com
HomePage          : https://github.com/mcc85s/FightingEntropy
StreetAddress     : 1718 US-9
City              : Clifton Park
State             : NY
PostalCode        : 12065
Country           : US
SamAccountName    : mcook85
UserPrincipalName : mcook85@securedigitsplus.com
ProfilePath       :
ScriptPath        :
HomeDirectory     :
HomeDrive         :
HomePhone         :
OfficePhone       : 518-406-8569
MobilePhone       : 518-406-8569
Fax               :
Title             : CEO/Security Engineer
Department        : Engineering
Company           : Secure Digits Plus LLC
Path              : OU=DevOps,DC=securedigitsplus,DC=com
Enabled           : 0
Exists            : 1
DistinguishedName : CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com

PS Prompt:\>
```

```
So, looks like a lot of information is there now.
The fields for ProfilePath, ScriptPath, HomeDirectory, and HomeDrive are all blank, but that may be simply set
to those values because there's no need to have those fields enabled quite yet in this example.

The rest of the information seems to indicate that this guys:
-Title      “CEO/Security Engineer”
-Department “Engineering”
-Company    “Secure Digits Plus LLC”
That all sounds pretty intense... dude's probably not playing with Lego's or anything like that.
```

```
    # Set [User.AccountPassword]
    $User.SetAccountPassword($Password)
```

```
PS Prompt:\> $Password
System.Security.SecureString
PS Prompt:\> $User.SetAccountPassword($Password)
VERBOSE: Performing the operation "Set-ADAccountPassword" on target "CN=Michael C.
Cook,OU=DevOps,DC=securedigitsplus,DC=com".
VERBOSE: Performing the operation "Set" on target "CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> $User

Name              : Michael C. Cook
```

```
DisplayName       : Michael C. Cook
GivenName         : Michael
Initials          : C
Surname           : Cook
Description       : Beginning the fight against ID theft and cybercrime
Office            : <Unspecified>
EmailAddress      : michael.c.cook.85@gmail.com
HomePage          : https://github.com/mcc85s/FightingEntropy
StreetAddress     : 1718 US-9
City              : Clifton Park
State             : NY
PostalCode        : 12065
Country           : US
SamAccountName    : mcook85
UserPrincipalName : mcook85@securedigitsplus.com
ProfilePath       :
ScriptPath        :
HomeDirectory     :
HomeDrive         :
HomePhone         :
OfficePhone       : 518-406-8569
MobilePhone       : 518-406-8569
Fax               :
Title             : CEO/Security Engineer
Department        : Engineering
Company           : Secure Digits Plus LLC
Path              : OU=DevOps,DC=securedigitsplus,DC=com
Enabled           : 1
Exists            : 1
DistinguishedName : CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com

PS Prompt:\>
```

```
And NOW, the user's account is enabled.
Still gotta add this user to a group, and then set the primary group for this user.
```

```powershell
    # Add user to group
    $Ctrl.AddAdGroupMember($Group,$User)
    # Set user primary group
    $User.SetPrimaryGroup($Group)
```

```
PS Prompt:\> # Add user to group
>> $Ctrl.AddAdGroupMember($Group,$User)
VERBOSE: Performing the operation "Set" on target "CN=Engineering,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\> # Set user primary group
>> $User.SetPrimaryGroup($Group)
VERBOSE: Performing the operation "Set" on target "CN=Michael C. Cook,OU=DevOps,DC=securedigitsplus,DC=com".
PS Prompt:\>
```

```
                                                                                                         ________/
\_______________________________________________________________________________________________________________/ Output
  Conclusion /---------------------------------------------------------------------------------------------------------\
/------------

    And there ya have it. That's basically it. This user should have access to the domain now, and they should be
    able to perform additional services or rolesso that additional users, computers, domain controllers, and et
    cetera can be created and deployed.

    This is actually necessary in order to use the other functions of the “Get-FEDCPromo” utility.
                                                                                                     ____________/
\___________________________________________________________________________________________________/ Conclusion

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