PowerShell – DSC with Linux & JEA:

1. Overview of PowerShell DSC for Linux:

What is PowerShell DSC for Linux?

-Desired State Configuration for Linux

-Enables consistent deployments of Linux assets

-CentOS

-Debian GNU/Linux

-Oracle Linux

-Red Hat Enterprise Linux

-SUSE Linux Enterprise Server

-Ubuntu Server

-nxArchive

-Archive files (.tar, .zip) are extracted whenever they are updated

Tools and resources available DSC for Linux is installed

-nxEnvironment

-Manage environment variables

-nxFile

-Manage files & directories

-nxFileLine

-Ensure that a file contains a specific line and/or does not contain lines matching a given pattern

-nxGroup

-Manage local Linux user groups

nxUser

-Manage local Linux users

-nxPackage

-Manage the installation state for Linux packages

-nxScript

-Deploy custom scripts

-nxService #Startup, shutdown, processes services

-Manage daemons (services)

-nxSshAuthorizedKeys #SecureShell Authorized Keys

-Manage the defined ssh Authorized (public) keys for a user account

1. Configuring a Linux Environment for DSC:

-Install Linux OS

-Visit <https://github.com/Microsoft/PowerShell-DSC-for-Linux>

-Check Supported Linux operating systems

1. CentOS 5, 6, and 7 (x86/x64)

2. Debian GNU/Linux 6, 7, and 8 (x86/x64)

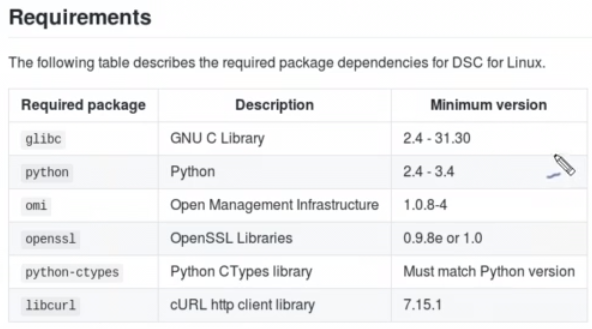
3. Oracle Linux 5, 6, and 7 (x86/x64)

4. Red Hat Enterprise Linux Server 5, 6 and 7

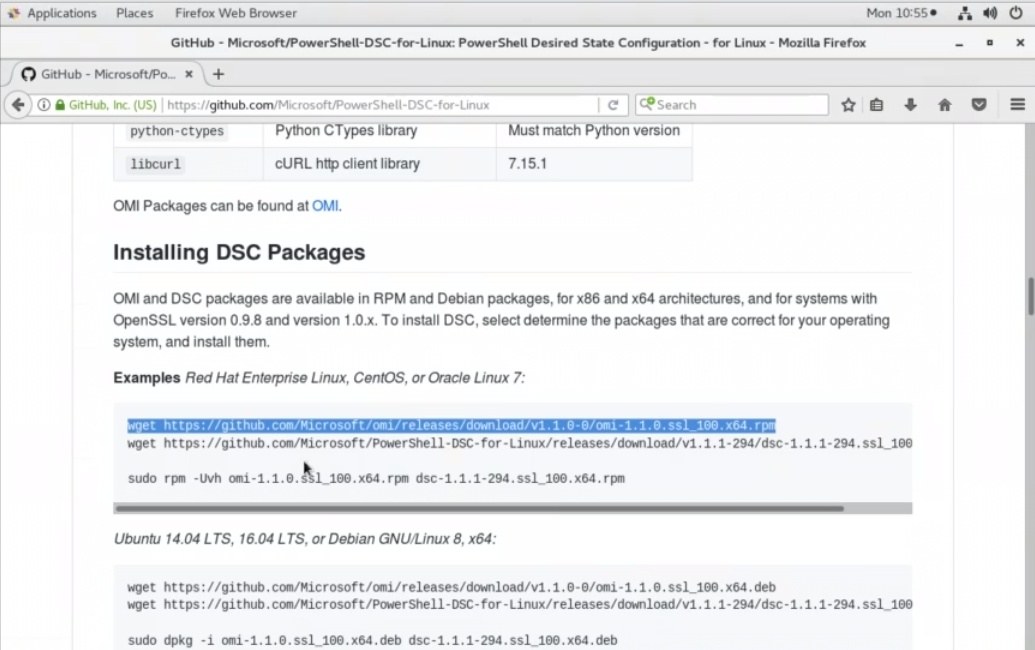
5. SUSE Linux Enterprise Server 10, 11, and 12

6. Ubuntu Server 12.04 LTS, 14.04 LTS, 16.04 LTS

-Check package pre-requisites:



Omi (Open Management Infrastructure) – Located on Github

-Can get DSC packages using wget

-Go to Linux terminal window

->

sudo wget wget

<https://github.com/Microsoft/omi/releases/download/v1.1.0-0/omi->1.1.0.ssl\_100.x64.deb

[sudo] password for administrator: Enter password will download all resources from the package

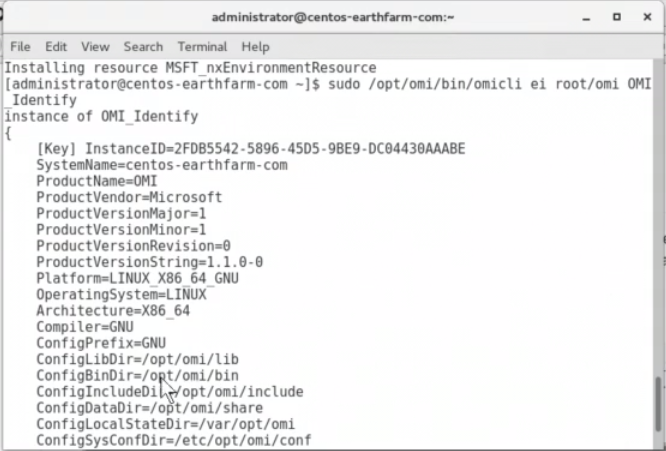
sudo wget

<https://github.com/Microsoft/PowerShell-DSC-forLinux/releases/download/v1.1.1-294/dsc-1.1.1-294.ssl_100.x64.deb>

sudo dpkg -i omi-1.1.0.ssl\_100.x64.deb dsc-1.1.1-294.ssl\_100.x64.deb

-Check omi configuration to validate changes of Linux

Sudo /opt/omi/bin/omicli ei root/omi OMI\_Identify



1. Creating a DSC Configuration for a Linux Node:

-Go to <https://www.microsoft.com/en-us/download/defaults.aspx?id=49150>

-Download & install PSDSCForLinux.msi

-Go to C:\Program Files(x86)\Microsoft PowerShell DSC for Linux\Modules\nx.zip

-Open Readme.txt

Copy everything in this folder into

C:\Windows\System32\WindowsPowerShell\v1.0\Modules\nx\

-nx.zip Extract all

-PowerShell ISE

#<#SampleFileConfig.ps1#>

Configuration SampleFileConfig

{

Import-DscResource -Module nx

$node = "CentOS.earthfarm.com"

Node $node

{

nxFile DirectoryExample

{

Ensure = "present"

DestinationPath = "/opt/mydir"

Type = "directory"

}

nxfile FileExample

{

Ensure = "present"

DestinationPath = "/opt/mydir/myfile"

#Content = @"#!/bin/bash`necho "hello world" `n"@

Content = "hello world `n"

Mode = "777" #Full permission

DependsOn = "[nxFile]DirectoryExample"

}

}

}

SampleFileConfig -OutputPath:"C:\temp"

1. Publishing a Linux DSC Config:

-Open Terminal window

Cd /opt

Ls

Cd Microsoft/

Cd dsc/

Ls

Bin etc keys lib module packages modules mof Scripts

Cd Scripts

Ls

Sudo ./GetDscConfiguration.py

-Haven’t config anything yet

-Go back to Windows

#Cim Session

#<#CimSession.ps1#>

$node = "CentOS.earthfarm.com"

$Credentials = Get-Credential -UserName:"Root" -Message:"Root Password Please:"

$opt = New-CimSessionOption -UseSsl:$true -SkipCACheck:$true -SkipCNCheck:$true -SkipRevocationCheck:$true

$sess = New-CimSession -Credential $Credentials -ComputerName: $node -Port:5986 -Authentication:basic -SessionOption:$opt -OperationTimeoutSec:90

#Id: 3

#Name: CimSession3

#InstanceId:

#ComputerName: CentOS.earthfarm.com

#Protocol: WSMAN

-Go to PowerShell

#PowerShell

#Start-DscConfiguration -Path:"C:\temp" -CimSession:$Sess -Wait -Verbose

#Linux Terminal

#sudo ./GetDscConfiguration.py

1. Examining a Linux Machine’s Config and Metaconfig:

-Linux Terminal

->

Cd /opt/Microsoft/dsc/

Ls

Cd Scripts

Sudo ./GetDscConfiguration.py

-Will prompt:

Instance of GetConfiguration

{

ReturnValue=0

Configurations=

{

[Key] DestinationPath=/opt/mydir

SourcePath=

Ensure=present

Type=directory

Force=false

Contents=[]

Checksum=

Recurse=false

Links=follow

Group=root

Mode=755

Owner=root

ModifiedDate=20171030193209.000000+480

}

Ls

Suo ./GetDscLocalConfigurationManager.py

#Configuration to run the configuration

#Associate with a Pull Server

-Go to Windows PowerShell[ise]

#<#SampleLinuxMetaConfig

[DscLocalConfigurationManager()]

Configuration SampleLinuxMetaConfig

{

Node "mylinuxserver"

{

Settings

{

RefreshFrequencyMins = 30;

RefreshMode = "PULL";

ConfigurationMode = "ApplyAndMonitor";

AllowModuleOverwrite = $true;

RebootNodeIfNeeded = $true;

ConfigurationModeFrequencyMins = 60;

}

ConfigurationRepositoryWeb ConfigurationManager

{

ServerURL = "https://PullServerMachine:8080/psdscpullserver.svc"

RegistryKey = "140a952b-b9d6-406b-b416-e0f759c9c0e4"

ConfigurationNames = @("MySQLRole")

}

}

}

SampleLinuxMetaConfig -OutputPath C:\temp\meta\

$Node = "CentOS.earthfarm.com"

$Credentials = Get-Credential -UserName:"root" -Message:"Enter Password:"

#F8 to run this to enter Credentials

$opt = New-CimSessionOption -UseSsl:$true -SkipCACheck:$true -SkipCNCheck:$true -SkipRevocationCheck:$true

$Sess = New-CimSession -Credential:$Credentials -ComputerName:$Node -Port:5986 -Authentication:basic -SessionOption:$opt -OperationTimeoutSec:90

#F8 to run this to enter Credentials

#F8 to run this to set

Set-DscLocalConfigurationManager -CimSession $Sess -Path c:\temp\meta –Verbose

-Go to Linux Terminal

Sudo ./GetDscLocalConfigurationManager.py

Cd ..

Cd ..

Ls

Dsc

Cd ..

Ls

Files Microsoft mydir omi rh

Cd omi

Ls

Bin lib LICENSE share

Cd bin

Ls

ConsistencyInvoker omicheck omiconfigeditor omireg service\_control

Omiagent omicli omigen

Cd ..

Cd .

Cd ./

Cd ..

Cd ..

Cd var

Cd opt

Cd omi

Ls

Lib log omiauth

Cd log

Ls

Vir dsc.log

#Show Log Event

#Log Event path

administrator@node:/var/opt/omi/log

1. Overview of JEA:

Why use Just Enough Administration(JEA)?

-Users with Admin rights pose a security risk

-It is not easy to remove admin rights

-JEA is a security technology

-Uses the principle of Least Privilege

-Users only have the privileges that are essential to their function

e.g. A user that is only meant to read documents should not be able to install software

-Benefits:

-Increased system stability

-Increased system security

-Easier deployment

-Reduce the numbers of administrators

-Virtual accounts

-Group managed service accounts

-Limit what users can do

-Cmdlets

-Functions

-External commands

-Understand what users are doing

-Transcripts

-Logs

1. Create a JEA Endpoint:

#<#131064.ps1#>

Install-Module xJEA

# Use this to check EndPoints

#After creating a DemoEP1 EndPoint

Get-PSSessionConfiguration

#Enter to a PS session on local computer with Demo1EP config

Enter-PSSession -ComputerName localhost -ConfigurationName Demo1EP

#To validate JEA settings

Stop-Process -Name notepad

#To validate JEA settings

Stop-Process -Name ServerManager

Exit-PSSession -ComputerName localhost

#C:\Program Files\WindowsPowerShell\Modules\xJea\0.2.16.6\Examples

#Find Demo1.ps1 & SetupJea.ps1

#<#SetupJea.ps1#>

Configuration SetupJea

{

Import-DscResource -ModuleName xJea

Node localhost

{

xJeaEndPoint CleanAll

{

Name = 'CleanAll'

CleanAll = $true

}

LocalConfigurationManager

{

RefreshFrequencyMins = 30

ConfigurationMode = "ApplyAndAutoCorrect"

DebugMode = "ForceModuleImport" #This disables #provider caching

}

}

}

SetupJea -OutputPath C:\JeaDemo

Set-DscLocalConfigurationManager -Path C:\JeaDemo -Verbose

Start-DscConfiguration -Path C:\JeaDemo -Wait -Verbose

#EOF

cls

#<#Demo1.ps1#>

Configuration Demo1

{

Import-DscResource -ModuleName xjea

xJeaToolKit Process

{

Name = 'Process'

# .csv list of command specifications in the order: 1.Name, #2.Parameter etc.

CommandSpecs = @"

Name, Parameter, ValidateSet, ValidatePattern

Get-Process

Get-Service

Stop-Process,Name,calc;notepad

Restart-Service,Name,,^A

"@

}

xJeaEndPoint Demo1EP

{

Name = 'Demo1EP'

Toolkit = 'Process'

SecurityDescriptorSddl =` 'O:NSG:BAD:P(A;;GX;;;WD)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)'

DependsOn = '[xJeaToolkit]Process'

}

}

Demo1 -OutputPath C:\JeaDemo

Set-DscLocalConfigurationManager -Path C:\JeaDemo -Verbose

Start-DscConfiguration -Path C:\JeaDemo -ComputerName localhost`

-Verbose -Wait -Debug -ErrorAction SilentlyContinue –ErrorVariable` $errors

if($errors | ? FullyQualifiedErrorId -ne 'HRESULT 0x803381fa')

{

$errors | Write-Error

}

Start-sleep -Seconds 30 #Wait for WINRM to restart

$s = New-PSSession -ComputerName . -ConfigurationName Demo1EP

icm $s {get-command} | out-string

icm $s {get-command stop-process -Syntax}

# Enter-PSSession $s

Remove-PSSession $s

#EOF

1. Testing a JEA Endpoint:

Same as section7, once connected to an endpoint, try running stop-process to different processes e.g. paint.exe

1. Resetting a JEA Endpoint:

Create an EndPoint configuration

#<#SetupJea.ps1#>

Configuration SetupJea

{

Import-DscResource -ModuleName xjea

Node localhost

{

xJeaEndPoint CleanAll

{

Name = 'CleanALL'

CleanAll = $true

}

LocalConfigurationManager

{

RefreshFrequencyMins = 30

ConfigurationMode = "ApplyAndAutoCorrect"

DebugMode = "ForceModuleImport" #This disables provider caching

}

}

}

SetupJea -OutputPath C:\JeaDemo

Set-DscLocalConfigurationManager -Path C:\JeaDemo -Verbose

Reset all EndPoint configuration

#<#131066.ps1#>

Get-PSSessionConfiguration

Configuration Clean

{

Import-DscResource -ModuleName xjea

Node localhost

{

xJeaEndPoint CleanAll

{

Name = "CleanAll"

CleanAll = $true

}

}

}

$mofPath = "C:\dsc"

Clean -OutputPath $mofPath

Start-DscConfiguration -Path $mofPath -Wait -Verbose -Force

Get-PSSessionConfiguration

1. Using the JEA Toolkit Helper:

JEA Helper Tool 2.0

-Create or Edit Role Capability

-Role Capabilities Design

-Configurations Listing, Mapping and Testing

-SDDL Helper

Example1:

* Go to Role Capabilities Design
* In row 2: Pick a cmdlet
* Select: Add-Printer
* Select: PortName
* Press: Add to Grid
* You’ll find from Bottom textbar to copy&paste
* VisibleCmdlets=
* VisibleFunctions=’Add-Printer’

Example2:

* Go to Configuration Listing, Mapping and Testing tab
* On right column
* Select a session configuration listed on the left column
* Press: Add Row
* Can add User or a Group

Example3:

* Go to SDDL Helper tab
* 2-step authentication
* Press Display SDDL
* Generate SDDL

#Unblock file to avoid error

#Unblock-File -Path C:\Users\phoenixyork\Desktop\JEAToolkitHelper.ps1

########################################################################################

# JEA Toolkit Helper

# Version 1.0

# by the Windows Server and System Center CAT team - http://aka.ms/bcb

# Please send feedback to brunosa@microsoft.com

########################################################################################

param (

[String]$SMAEndpointWS = "",

[String]$SMAEndpointPort = "9090"

)

$ToolVersion = "1.0"

$Global:DefaultSDDL = "O:NSG:BAD:P(A;;GA;;;BA)(A;;GA;;;RM)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)"

################################################

# Functions

################################################

function Popup()

{

param (

[String]$Message

)

$a = new-object -comobject wscript.shell

$b = $a.popup($Message,0,"JEA Toolkit Helper",0)

}

Function AddArray()

{

param (

[String]$Module,

[String]$Name,

[String]$Parameter,

[String]$ValidateSet = "",

[String]$ValidatePattern = ""

)

$Global:CommandArray = @()

$tmpObject = select-object -inputobject "" IsChecked, Module, Name, Parameter, ValidateSet, ValidatePattern

$tmpObject.Ischecked = $false

$tmpObject.Module = $Module

$tmpObject.Name = $Name

$tmpObject.Parameter = $Parameter

$tmpObject.ValidateSet = $ValidateSet

$tmpObject.ValidatePattern = $ValidatePattern

$Global:CommandArray += $tmpObject

If ($FORM.FindName('CSVGrid').Items.Count -eq 0)

{$FORM.FindName('CSVGrid').ItemsSource = $Global:CommandArray}

else {$FORM.FindName('CSVGrid').ItemsSource += $Global:CommandArray}

}

Function UpdateDelegation()

{

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Validating account list to generate SDDL string in script output..."

$Global:SDDLAccountList = @()

$Accounts = $FORM.FindName('ConfigureAllowedUsersTextBox').Text.Split(";")

$Global:strSDDL = "O:NSG:BAD:P"

$ErrorAccounts = 0

foreach ($Account in $Accounts)

{

$objUser = New-Object System.Security.Principal.NTAccount($Account)

$eap = $ErrorActionPreference = "SilentlyContinue"

$strSID = $objUser.Translate([System.Security.Principal.SecurityIdentifier])

if (!$?) {

$ErrorActionPreference =$eap

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : SID could not be resolved for account $Account. This account has not been added to the resulting SDDL output."

$ErrorAccounts += 1

}

else

{

$Global:strSDDL = $Global:strSDDL + "(A;;GA;;;" + $strSID + ")"

$Global:SDDLAccountList +=$Account

}

}

$Global:strSDDL = $Global:strSDDL + "S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)"

If ($ErrorAccounts -eq $Accounts.Count)

{

$Global:strSDDL = $Global:DefaultSDDL

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : None of the user(s) or group(s) provided could be resolved, so the JEA Toolkit will use the default security."

}

UpdateScriptOutput

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Validating account list to generate SDDL string in script output...done!"

}

Function UpdateCmdletList()

{

$FORM.FindName('PickCmdletComboBox').Items.Clear()

Foreach ($CmdletItem in $Global:CmdletList)

{ If ($CmdletItem.Name.Length -gt 3) {$FORM.FindName('PickCmdletComboBox').Items.Add($CmdletItem.Name) | out-null} }

}

Function UpdateModuleList()

{

$FORM.FindName('FilterModuleComboBox').Items.Clear()

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Loading modules list...This may take a few seconds, please wait..."

#$ModuleList = Get-Module | Sort-Object Name | Select Name

$ModuleList = Get-Module -ListAvailable | Sort-Object Name | Select Name

Foreach ($ModuleItem in $ModuleList)

{ If ($ModuleList.Name.Length -gt 3) {$FORM.FindName('FilterModuleComboBox').Items.Add($ModuleItem.Name) | out-null} }

}

Function UpdateScriptOutput()

{

If ($FORM.FindName('CSVGrid').ItemsSource.Count -gt 0)

{

$OriginalScript = @"

configuration TOOLKITNAMEPLACEHOLDER

{

Import-DscResource -module xjea

xJeaToolKit TOOLKITNAMEPLACEHOLDER

{

Name = 'TOOLKITNAMEPLACEHOLDER'

CommandSpecs = @"

Module, Name,Parameter,ValidateSet,ValidatePattern

COMMANDSPLACEHOLDER

"`@

# Or you can use this command, replacing it with the right file and path names

# CommandSpecs = Get-Content “¡±C:\AuditorToolkit\Toolkit.csv”¡L –VDelimiter “¡±NoSuch”¡L

}

xJeaEndPoint TOOLKITNAMEPLACEHOLDEREP

{

Name = 'TOOLKITNAMEPLACEHOLDEREP'

Toolkit = 'TOOLKITNAMEPLACEHOLDER'

SecurityDescriptorSddl = 'SDDLSTRINGPLACEHOLDER'

DependsOn = '[xJeaToolKit]TOOLKITNAMEPLACEHOLDER'

#CleanAll = 'True'

}

}

SDDLDESCRIPTIONPLACEHOLDER

#The first two lines of the script below are enabled when importing the toolkit,

#and would have to be enabled in case of manual import as well

#Note that these same lines are also used for removal

#(in which case the script enabled the 'CleanAll' property in the EndPoint configuration)

#The WinRM service restart is not done by the script, but you could chose to add for manual execution

#TOOLKITNAMEPLACEHOLDER -OutputPath C:\DSCDemo

#Start-DscConfiguration -Path C:\DSCDemo -ComputerName localhost -Verbose -wait -debug

#start-sleep -Seconds 30

#The next three lines are the only ones executed when testing a toolkit on the local computer.

#This is also what you would run, should you want to do this manually

#`$s = New-PSSession -ComputerName . -ConfigurationName TOOLKITNAMEPLACEHOLDEREP

#Invoke-command `$s {get-command} |out-string

#Remove-PSSession `$s

#Alternatively, once a new session has been created, this is how you could enter

#and work in the JEA session before exiting and removing it.

#Enter-pssession `$s

#Exit-PSSession

"@

$NewScriptContent = ""

Foreach ($CurrentRow in $FORM.FindName('CSVGrid').ItemsSource)

{

If ($CurrentRow.Module -eq $null)

{$NewScriptContent+= "," + $CurrentRow.Name.Trim() + "," + $CurrentRow.Parameter + "," + $CurrentRow.ValidateSet + "," + $CurrentRow.ValidatePattern + "`r`n"}

else

{$NewScriptContent+= $CurrentRow.Module.Trim() + "," + $CurrentRow.Name.Trim() + "," + $CurrentRow.Parameter + "," + $CurrentRow.ValidateSet + "," + $CurrentRow.ValidatePattern + "`r`n"}

}

$FORM.FindName('ScriptOutputTextBlock').Text=$OriginalScript.Replace("COMMANDSPLACEHOLDER", $NewScriptContent)

$FORM.FindName('ScriptOutputTextBlock').Text=$FORM.FindName('ScriptOutputTextBlock').Text.Replace("TOOLKITNAMEPLACEHOLDER", $FORM.FindName('ToolkitNameTextBox').Text)

$FORM.FindName('ScriptOutputTextBlock').Text=$FORM.FindName('ScriptOutputTextBlock').Text.Replace("SDDLSTRINGPLACEHOLDER", $Global:strSDDL)

If ($Global:strSDDL -eq $Global:DefaultSDDL)

{

$FORM.FindName('ScriptOutputTextBlock').Text=$FORM.FindName('ScriptOutputTextBlock').Text.Replace("SDDLDESCRIPTIONPLACEHOLDER", "#SDDL Description : Default SDDL (BULTIN\Administrators group on the target endpoint)")

}

else

{

$SDDLDescriptionOutput = "#SDDL Description :"

foreach ($SDDLAccount in $Global:SDDLAccountList) {$SDDLDescriptionOutput = $SDDLDescriptionOutput + $SDDLAccount + ";"}

$FORM.FindName('ScriptOutputTextBlock').Text=$FORM.FindName('ScriptOutputTextBlock').Text.Replace("SDDLDESCRIPTIONPLACEHOLDER", $SDDLDescriptionOutput)

}

$FORM.FindName('ImportToolkit').IsEnabled = $true

$FORM.FindName('TestToolkit').IsEnabled = $true

$FORM.FindName('RemoveToolkit').IsEnabled = $true

$FORM.FindName('BypassSMACmdletsNotPresentCB').IsEnabled = $true

$FORM.FindName('BypassSMACmdletsNotPresentLabel').IsEnabled = $true

}

else

{

$FORM.FindName('ScriptOutputTextBlock').Text="Script output will be updated here"

$FORM.FindName('ImportToolkit').IsEnabled = $false

$FORM.FindName('TestToolkit').IsEnabled = $false

$FORM.FindName('RemoveToolkit').IsEnabled = $false

$FORM.FindName('BypassSMACmdletsNotPresentCB').IsEnabled = $false

$FORM.FindName('BypassSMACmdletsNotPresentLabel').IsEnabled = $false

}

}

################################################

# Form definition

################################################

Add-Type -AssemblyName PresentationFramework

Add-Type -AssemblyName System.Windows.Forms

[XML]$XAML = @'

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

ResizeMode="NoResize"

Title="JEA Toolkit Helper" Height="695" Width="840">

<Window.Resources>

<Style TargetType="{x:Type TabItem}">

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type TabItem}">

<Grid>

<Border Name="Border" Background="LightBlue" BorderBrush="Black" BorderThickness="1,1,1,0" CornerRadius="25,25,0,0" >

<ContentPresenter x:Name="ContentSite" VerticalAlignment="Center" HorizontalAlignment="Center" ContentSource="Header" Margin="12,2,12,2"/>

</Border>

</Grid>

<ControlTemplate.Triggers>

<Trigger Property="IsSelected" Value="True">

<Setter TargetName="Border" Property="Background" Value="LightBlue" />

</Trigger>

<Trigger Property="IsSelected" Value="False">

<Setter TargetName="Border" Property="Background" Value="White" />

</Trigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</Window.Resources>

<Grid>

<TabControl>

<TabItem>

<TabItem.Header>

<StackPanel Orientation="Horizontal">

<TextBlock Text="Design Helper" Margin="2,0,0,0" VerticalAlignment="Center" />

</StackPanel>

</TabItem.Header>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="30"/>

<RowDefinition Height="45"/>

<RowDefinition Height="45"/>

<RowDefinition Height="40"/>

<RowDefinition Height="40"/>

<RowDefinition Height="40"/>

<RowDefinition Height="320"/>

<RowDefinition Height="30"/>

<RowDefinition Height="30"/>

<RowDefinition Height="30"/>

</Grid.RowDefinitions>

<Label FontWeight="Bold" Content="We are working with Toolkit named " HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5,0,0,0" Grid.Row="0"></Label>

<TextBox Text="DemoXYZ" Name="ToolkitNameTextBox" IsEnabled="True" Grid.Row="0" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="230,0,0,0" Width="100"></TextBox>

<Label Content="You can import an existing CSV file" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="5,0,0,0" Grid.Row="1"></Label>

<Button Content="Import CSV File..." Name="ImportCSVFile" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="230,0,0,0" Width="120" Grid.Row="1"/>

<ComboBox Name="ImportCSVFileAction" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="380,0,0,0" Width="150" Grid.Row="1"></ComboBox>

<CheckBox Name="ImportXMLCB" IsChecked="True" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="550,0,0,0" Grid.Row="1"></CheckBox>

<Label Name="ImportXMLLabel" Content="Import delegation data as well, if available" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="570,0,0,0" Grid.Row="1"></Label>

<Label HorizontalAlignment="Left" Margin="5,0,0,0" MaxWidth="170" Grid.Row="2">

<TextBlock Name="PickCmdletLabel" Text="Or you can pick a cmdlet and - optionally - properties" TextWrapping= "Wrap"></TextBlock>

</Label>

<ComboBox Name="PickCmdletComboBox" IsEditable="False" HorizontalAlignment="Left" VerticalAlignment="Center" Width="215" Margin="230,0,0,0" Grid.Row="2"></ComboBox>

<ComboBox Name="PickPropertiesComboBox" HorizontalAlignment="Left" VerticalAlignment="Center" Width="215" Margin="475,0,0,0" Grid.Row="2">

<ComboBox.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">

<CheckBox Margin="5" IsChecked="{Binding PropertyChecked}"/>

<TextBlock Margin="5" Text="{Binding PropertyName}"/>

</StackPanel>

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<Button Content="Add to Toolkit" Name="AddToGrid" VerticalAlignment="Center" HorizontalAlignment="Left" Margin="710,0,0,0" Width="100" Grid.Row="2"/>

<Label HorizontalAlignment="Left" Margin="5,0,0,0" MaxWidth="210" Grid.Row="3" Grid.RowSpan="2">

<TextBlock Name="PickModuleLabel" Text="Or you can add a full/partial module, or use it to filter the cmdlets list" TextWrapping= "Wrap"></TextBlock>

</Label>

<ComboBox Name="FilterModuleComboBox" IsEditable="False" HorizontalAlignment="Left" VerticalAlignment="Center" Width="120" Margin="230,0,0,0" Grid.Row="3"></ComboBox>

<Button Content="Add to Toolkit" Name="AddModuleFullToGridButton" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="365,0,0,0" Width="100" Grid.Row="3"/>

<Button Content="Add Get-\* only" Name="AddModuleGetToGridButton" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="480,0,0,0" Width="100" Grid.Row="3"/>

<Button Content="Filter Cmdlets" Name="FilterModuleButton" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="595,0,0,0" Width="100" Grid.Row="3"/>

<Button Content="Remove Filter" Name="RemoveFilterModuleButton" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="710,0,0,0" Width="100" Grid.Row="3"/>

<Label Content="Module to import" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="230,0,0,0" Grid.Row="4"></Label>

<TextBox Text="" Name="ImportModuleTextBox" HorizontalAlignment="Left" VerticalAlignment="Center" Width="215" Margin="365,0,0,0" Grid.Row="4"></TextBox>

<Button Content="Import Module" Name="ImportModuleButton" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="595,0,0,0" Width="100" Grid.Row="4"/>

<Label Name="PickRunbookLabelContainer" VerticalAlignment="Center" HorizontalAlignment="Left" Margin="5,0,0,0" MaxWidth="230" Grid.Row="5">

<TextBlock Name="PickRunbookLabel" Text="Or you can pick SMA Runbook(s)" TextWrapping= "Wrap"></TextBlock>

</Label>

<ComboBox Name="PickRunbooksComboBox" HorizontalAlignment="Left" VerticalAlignment="Center" Width="460" Margin="230,0,0,0" Grid.Row="5">

<ComboBox.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">

<CheckBox Margin="5" IsChecked="{Binding RunbookChecked}"/>

<TextBlock Margin="5" Text="{Binding RunbookName}"/>

</StackPanel>

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<Button Content="Add to Toolkit" Name="AddRunbookToGrid" VerticalAlignment="Center" HorizontalAlignment="Left" Margin="710,0,0,0" Width="100" Grid.Row="5"/>

<DataGrid AutoGenerateColumns="False" Margin="10,0,0,0" Name="CSVGrid" HorizontalAlignment="Left" VerticalAlignment="Top" Height="320" Width="800" ItemsSource="{Binding}" SelectionUnit="Cell" Grid.Row="6">

<DataGrid.Columns>

<DataGridCheckBoxColumn Binding="{Binding Path=IsChecked}"/>

<DataGridTextColumn Binding="{Binding Path=Module}" Header="Module"/>

<DataGridTextColumn Binding="{Binding Path=Name}" Header="Name"/>

<DataGridTextColumn Binding="{Binding Path=Parameter}" Header="Parameter" />

<DataGridTextColumn Binding="{Binding Path=ValidateSet}">

<DataGridTextColumn.Header>

<TextBlock Text="ValidateSet" ToolTipService.ToolTip="Semi-colon separated list of allowed parameters. An empty list means that all parameters are allowed." />

</DataGridTextColumn.Header>

<DataGridTextColumn.ElementStyle>

<Style TargetType="{x:Type TextBlock}">

<Setter Property="ToolTip" Value="{Binding Description}" />

<Setter Property="TextWrapping" Value="Wrap" />

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

<DataGridTextColumn Binding="{Binding Path=ValidatePattern}">

<DataGridTextColumn.Header>

<TextBlock Text="ValidatePattern" ToolTipService.ToolTip="A regular expression. This is an optional parameter. See examples here : http://technet.microsoft.com/en-us/library/hh847880.aspx" />

</DataGridTextColumn.Header>

<DataGridTextColumn.ElementStyle>

<Style TargetType="{x:Type TextBlock}">

<Setter Property="ToolTip" Value="{Binding Description}" />

<Setter Property="TextWrapping" Value="Wrap" />

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

</DataGrid.Columns>

</DataGrid>

<Button Content="Add Row" Name="AddRowGrid" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="10,0,0,0" Width="150" Grid.Row="7"/>

<Button Content="Remove Selected Row(s)" Name="DeleteRowGrid" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="170,0,0,0" Width="150" Grid.Row="7"/>

<Button Content="Remove All Rows" Name="DeleteAllRows" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="330,0,0,0" Width="150" Grid.Row="7"/>

<CheckBox Name="ConfigureAllowedUsersCB" IsChecked="False" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="10,0,0,0" Grid.Row="8"></CheckBox>

<Label Name="ConfigureAllowedUsersLabel" Content="Configure Allowed Users (default is BUILTIN\Administrators)" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="30,0,0,0" Grid.Row="8"></Label>

<TextBox Text="" Name="ConfigureAllowedUsersTextBox" IsEnabled="False" Grid.Row="8" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="370,0,0,0" Width="280"></TextBox>

<Button Content="Update Delegation" Name="UpdateDelegation" IsEnabled="False" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="660,0,0,0" Width="150" Grid.Row="8"/>

</Grid>

</TabItem>

<TabItem>

<TabItem.Header>

<StackPanel Orientation="Horizontal">

<TextBlock Text="Script output" Margin="2,0,0,0" VerticalAlignment="Center" />

</StackPanel>

</TabItem.Header>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="480"/>

<RowDefinition Height="40"/>

<RowDefinition Height="40"/>

<RowDefinition Height="40"/>

<RowDefinition Height="30"/>

</Grid.RowDefinitions>

<TextBox Name="ScriptOutputTextBlock" TextWrapping="Wrap" AcceptsReturn="True" VerticalScrollBarVisibility="Visible" Grid.Row="0">

Script output will be updated here

</TextBox>

<Label Content="You can copy and paste this script in PowerShell/ISE, or use the button on the right to copy it to the clipboard" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="10,0,0,0" Width="630" Grid.Row="1"></Label>

<Button Content="Copy to Clipboard" Name="CopyToClipboard" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="660,0,0,0" Width="150" Grid.Row="1"/>

<Label Content="You can also save this output as a CSV File Name" Name="CSVFileNameLabel" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="10,0,0,0" Grid.Row="2"></Label>

<TextBox Text="DemoXYZ.csv" Name="CSVFileNameTextBox" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="300,0,0,0" Width="150" Grid.Row="2"></TextBox>

<CheckBox Name="ExportXMLCB" IsChecked="True" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="460,0,0,0" Grid.Row="2"></CheckBox>

<Label Name="ExportXMLLabel" Content="Export delegation data as well" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="480,0,0,0" Grid.Row="2"></Label>

<Button Content="Export to CSV" Name="ExportCSVFile" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="660,0,0,0" Width="150" Grid.Row="2"/>

<Label Content="Finally, you can import, test or this configuration on the local machine" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="10,0,0,0" Width="630" Grid.Row="3"></Label>

<Button Content="Remove" Name="RemoveToolkit" IsEnabled="False" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="700,0,0,0" Width="100" Grid.Row="3"/>

<Button Content="Test" Name="TestToolkit" IsEnabled="False" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="550,0,0,0" Width="100" Grid.Row="3"/>

<Button Content="Import" Name="ImportToolkit" IsEnabled="False" Visibility="visible" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="400,0,0,0" Width="100" Grid.Row="3"/>

<CheckBox Name="BypassSMACmdletsNotPresentCB" IsEnabled="False" IsChecked="True" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="400,0,0,0" Grid.Row="4"></CheckBox>

<Label Name="BypassSMACmdletsNotPresentLabel" IsEnabled="False" Content="If SMA module not present, disable SMA cmdlets before toolkit import" HorizontalAlignment="Left" VerticalAlignment="Center" Margin="420,0,0,0" Grid.Row="4"></Label>

</Grid>

</TabItem>

</TabControl>

</Grid>

</Window>

'@

$Reader = (New-Object System.XML.XMLNodeReader $XAML)

$FORM = [Windows.Markup.XAMLReader]::Load($Reader)

################################################

# Events

################################################

$FORM.FindName('AddRowGrid').Add\_Click({

#write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Adding row..."

AddArray -Name "" -Parameter ""

UpdateScriptOutput

})

$FORM.FindName('DeleteRowGrid').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Deleting items from grid..."

$Global:CommandArray = @()

$Global:CommandArray += $FORM.FindName('CSVGrid').Itemssource | ? IsChecked -eq $False

$FORM.FindName('CSVGrid').ItemsSource = $Global:CommandArray

UpdateScriptOutput

})

$FORM.FindName('DeleteAllRows').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Deleting all items from grid..."

$Global:CommandArray = @()

$FORM.FindName('CSVGrid').ItemsSource = $Global:CommandArray

UpdateScriptOutput

})

$FORM.FindName('AddModuleFullToGridButton').Add\_Click({

If ($FORM.FindName('FilterModuleComboBox').Text -eq "")

{

popup -Message "Please select a module"

}

else

{

If (($FORM.FindName('CSVGrid').ItemsSource | ? Module -eq $FORM.FindName('FilterModuleComboBox').Text | ? Name -eq "").Parameter.Count -eq 0)

{

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Adding items to grid..."

AddArray -Module $FORM.FindName('FilterModuleComboBox').Text -Name "" -Parameter "" -ValidateSet ""

UpdateScriptOutput

}

else

{

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Module is already in the grid, we're not adding it again..."

}

}

})

$FORM.FindName('AddModuleGetToGridButton').Add\_Click({

If ($FORM.FindName('FilterModuleComboBox').Text -eq "")

{

popup -Message "Please select a module"

}

else

{

If (($FORM.FindName('CSVGrid').ItemsSource | ? Module -eq $FORM.FindName('FilterModuleComboBox').Text | ? Name -eq "Get-\*").Parameter.Count -eq 0)

{

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Adding items to grid..."

AddArray -Module $FORM.FindName('FilterModuleComboBox').Text -Name "Get-\*" -Parameter "" -ValidateSet ""

UpdateScriptOutput

}

else

{

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Module is already in the grid with Get-\* cmdlets, we're not adding it again..."

}

}

})

$FORM.FindName('AddToGrid').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Adding items to grid..."

$PropertiesDiscarded = 0

$PropertiesAdded = 0

$PropertiesChecked = $false

Foreach ($Property in $FORM.FindName('PickPropertiesComboBox').Items)

{

If ($Property.PropertyChecked -eq $true)

{

$PropertiesChecked = $true

If (($FORM.FindName('CSVGrid').ItemsSource | ? Name -eq $FORM.FindName('PickCmdletComboBox').Text | ? Parameter -eq $Property.PropertyName).Parameter.Count -eq 0)

{

$command = (get-command $FORM.FindName('PickCmdletComboBox').Text)

$potentialvalues=@()

try {

If (($command.ResolveParameter($Property.PropertyName).ParameterType.Name) -eq "String")

{

$p=$command.Parametersets[0].parameters |?{$\_.name -eq $Property.PropertyName}

$potentialvalues = ($p.Attributes).ValidValues

# Thanks http://blogs.msdn.com/b/powershell/archive/2006/05/10/594175.aspx?Redirected=true

}

else

{

$potentialvalues = [Enum]::GetNames($command.ResolveParameter($Property.PropertyName).ParameterType.FullName)

}

}

catch {}

AddArray -Module "" -Name $FORM.FindName('PickCmdletComboBox').Text -Parameter $Property.PropertyName -ValidateSet ($potentialvalues -join ";")

$PropertiesAdded = $PropertiesAdded +1

}

else

{$PropertiesDiscarded = $PropertiesDiscarded +1}

}

}

If ($PropertiesAdded -eq 0)

{

If (($FORM.FindName('CSVGrid').ItemsSource | ? Name -eq $FORM.FindName('PickCmdletComboBox').Text).Parameter.Count -eq 0)

{

AddArray -Name $FORM.FindName('PickCmdletComboBox').Text -Parameter ""

}

else

{write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] NOTE : The cmdlet was not added, because it was already found in the grid."}

}

If ($PropertiesDiscarded -gt 0) {write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] NOTE : $PropertiesDiscarded propertie(s) were not added, because they were already found in the grid."}

#write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Adding items to grid...done!"

UpdateScriptOutput

})

$FORM.FindName('AddRunbookToGrid').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Adding Runbooks to grid..."

$RunbooksToAdd = ""

Foreach ($Property in $FORM.FindName('PickRunbooksComboBox').Items)

{

If ($Property.RunbookChecked -eq $true)

{

If (($FORM.FindName('CSVGrid').ItemsSource | ? Name -eq $FORM.FindName('PickRunbooksComboBox').Text | ? Parameter -eq $Property.PropertyName).Parameter.Count -eq 0)

{

$RunbooksToAdd += ";" + $Property.RunbookName

}

}

}

$RunbooksToAdd = $RunbooksToAdd.Substring(1,$RunbooksToAdd.Length-1)

#We check if there are already some entries for the SMA cmdlets and, if yes, we add to them

If (($FORM.FindName('CSVGrid').ItemsSource | ? Name -eq "Start-SmaRunbook" | ? Parameter -eq "Name").Parameter.Count -ne 0)

{

$RunbookToAdd = (($RunbooksToAdd + ";" + ($FORM.FindName('CSVGrid').ItemsSource | ? Name -eq "Start-SmaRunbook" | ? Parameter -eq "Name").ValidateSet).Split(";") | select-object -unique) -join ";"

$Global:CommandArray = @()

$Global:CommandArray += $FORM.FindName('CSVGrid').Itemssource | ? Name -ne "Start-SmaRunbook"

$FORM.FindName('CSVGrid').ItemsSource = $Global:CommandArray

}

AddArray -Module "" -Name "Start-SmaRunbook" -Parameter "Name" -ValidateSet $RunbooksToAdd

AddArray -Module "" -Name "Start-SmaRunbook" -Parameter "Parameters" -ValidateSet ""

AddArray -Module "" -Name "Start-SmaRunbook" -Parameter "WebServiceEndpoint" -ValidateSet $Global:SMAWS

AddArray -Module "" -Name "Start-SmaRunbook" -Parameter "Port" -ValidateSet $Global:SMAPort

#AddArray -Module "" -Name "Stop-SmaRunbook" -Parameter "Name" -ValidateSet $RunbooksToAdd

UpdateScriptOutput

})

$FORM.FindName('PickCmdletComboBox').Add\_DropDownClosed({

If ($FORM.FindName('PickCmdletComboBox').Text)

{

$SelectedCmdletParameters = (Get-Command $FORM.FindName('PickCmdletComboBox').Text | % parameters).keys

$Global:PropertyArray = @()

Foreach ($SelectedCmdletParameter in $SelectedCmdletParameters)

{

$tmpObject2 = select-object -inputobject "" PropertyChecked, PropertyName

$tmpObject2.PropertyChecked = $false

$tmpObject2.PropertyName = $SelectedCmdletParameter

$Global:PropertyArray += $tmpObject2

$FORM.FindName('PickPropertiesComboBox').ItemsSource = $Global:PropertyArray

$FORM.FindName('PickPropertiesComboBox').IsDropDownOpen = $true

}

}

})

$FORM.FindName('ImportCSVFile').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Importing..."

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Importing CSV File..."

[void] [System.Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")

$OpenFileWindow = New-Object System.Windows.Forms.OpenFileDialog

$OpenFileWindow.InitialDirectory = (Get-Location -PSProvider FileSystem).Path

$OpenFileWindow.ShowHelp=$false

$OpenFileWindow.Filter = "csv files (\*.csv)|\*.csv";

if($OpenFileWindow.ShowDialog() -eq "OK")

{

$CSVFileLocation = $OpenFileWindow.FileName.substring(0, $OpenFileWindow.FileName.LastIndexOf("\"))

$CSVFilName = ($OpenFileWindow.FileName.split("\")[$OpenFileWindow.FileName.split("\").Count-1]).split(".")[0]

$NewCSVData = Import-CSV $OpenFileWindow.FileName | Select-Object IsChecked, Module, Name, Parameter, ValidateSet, ValidatePattern

If ($FORM.FindName('ImportCSVFileAction').Text -eq "Replace grid content")

{$FORM.FindName('CSVGrid').ItemsSource = $NewCSVData}

else {$FORM.FindName('CSVGrid').ItemsSource += $NewCSVData}

Foreach ($Property in $FORM.FindName('CSVGrid').Itemssource) {$Property.IsChecked="False"}

$FORM.FindName('ToolkitNameTextBox').Text = $CSVFilName.split(".")[0]

UpdateScriptOutput

If (($FORM.FindName('ImportXMLCB').IsChecked) -and (Test-Path -Path ($CSVFileLocation + "\" + $FORM.FindName('ToolkitNameTextBox').Text + ".xml")))

{

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Importing delegation data file $XMLFile..."

$XmlReader= New-Object System.Xml.XmlTextReader(($CSVFileLocation + "\" + $FORM.FindName('ToolkitNameTextBox').Text + ".xml"))

While ($XmlReader.Read()){

If ($XmlReader.NodeType -eq [System.Xml.XmlNodeType]::Element){

switch ($XmlReader.Name){

"DelegationData" {$FORM.FindName('ConfigureAllowedUsersTextBox').Text=$XmlReader.ReadString()}

}

}

}

$XmlReader.Close()

If ($FORM.FindName('ConfigureAllowedUsersTextBox').Text)

{

$FORM.FindName('ConfigureAllowedUsersCB').IsChecked=$True

$FORM.FindName('ConfigureAllowedUsersTextBox').IsEnabled=$True

$FORM.FindName('UpdateDelegation').IsEnabled=$True

UpdateDelegation

}

}

else

{

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] No delegation data found or the option was not checked. The tool will not update the delegation text box."

}

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Importing...Done!"

}

else

{

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] No CSV file was specified by user, returning to main window..."

}

})

$FORM.FindName('ExportCSVFile').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Exporting..."

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Exporting to CSV File" $FORM.FindName('CSVFileNameTextBox').Text "..."

If ($FORM.FindName('CSVGrid').Itemssource)

{

$FORM.FindName('CSVGrid').Itemssource | export-csv ((Get-Location -PSProvider FileSystem).Path + "\" + $FORM.FindName('CSVFileNameTextBox').Text) -notypeinformation

popup -Message ("CSV file " + $FORM.FindName('CSVFileNameTextBox').Text + " was created in folder " + (Get-Location -PSProvider FileSystem).Path)

If ($FORM.FindName('ExportXMLCB').IsChecked)

{

$XMLFile = $FORM.FindName('CSVFileNameTextBox').Text.Substring(0, $FORM.FindName('CSVFileNameTextBox').Text.Length -4) + ".xml"

$AllowedUsers = $FORM.FindName('ConfigureAllowedUsersTextBox').Text

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Exporting delegation data to file $XMLFile..."

$XMLData =

@”¡L

<DefaultConfiguration>

<DelegationData>$AllowedUsers</DelegationData>

</DefaultConfiguration>

“¡±@

$XMLData | Out-File $XMLFile -Force

}

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Exporting...Done!"

}

else

{

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Grid is empty, file was not created."

popup -Message ("Grid is empty, file was not created.")

}

})

$FORM.FindName('FilterModuleButton').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Filtering cmdlet list for module" $FORM.FindName('FilterModuleComboBox').Text "..."

$Global:CmdletList = Get-Command -Module $FORM.FindName('FilterModuleComboBox').Text | Sort-Object Name | Select Name

UpdateCmdletList

})

$FORM.FindName('RemoveFilterModuleButton').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Removing filter on cmdlet list..."

$Global:CmdletList = Get-Command | Sort-Object Name | Select Name

UpdateCmdletList

})

$FORM.FindName('ImportModuleButton').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Importing module" $FORM.FindName('ImportModuleTextBox').Text "..."

$eap = $ErrorActionPreference = "SilentlyContinue"

Import-Module $FORM.FindName('ImportModuleTextBox').Text

if (!$?) {

$ErrorActionPreference =$eap

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : Module" $FORM.FindName('ImportModuleTextBox').Text "could not be imported. Please check module name and existence." }

popup -Message ("WARNING : Module " + $FORM.FindName('ImportModuleTextBox').Text + " could not be imported. Please check module name and existence.")

else{

$ErrorActionPreference =$eap

$Global:CmdletList = Get-Command | Sort-Object Name | Select Name

UpdateCmdletList

UpdateModuleList

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Importing module" $FORM.FindName('ImportModuleTextBox').Text "...Done!"

popup -Message ("Module " + $FORM.FindName('ImportModuleTextBox').Text + " was imported.")

}

})

$FORM.FindName('CopyToClipboard').Add\_Click({

$null = [Reflection.Assembly]::LoadWithPartialName(“¡±System.Windows.Forms”¡L)

$dataObject = New-Object windows.forms.dataobject

$dataObject.SetData([Windows.Forms.DataFormats]::UnicodeText, $true, $FORM.FindName('ScriptOutputTextBlock').Text)

[Windows.Forms.Clipboard]::SetDataObject($dataObject, $true)

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Copied script content to clipboard."

popup -Message "Script content was copied to clipboard."

})

$FORM.FindName('ConfigureAllowedUsersCB').Add\_Checked({

$FORM.FindName('ConfigureAllowedUsersTextBox').IsEnabled=$True

$FORM.FindName('UpdateDelegation').IsEnabled=$True

})

$FORM.FindName('ConfigureAllowedUsersCB').Add\_UnChecked({

$FORM.FindName('ConfigureAllowedUsersTextBox').IsEnabled=$False

$FORM.FindName('UpdateDelegation').IsEnabled=$false

})

$FORM.FindName('ToolkitNameTextBox').Add\_TextChanged({

$FORM.FindName('CSVFileNameTextBox').Text = $FORM.FindName('ToolkitNameTextBox').Text + ".csv"

UpdateScriptOutput

})

$FORM.FindName('UpdateDelegation').Add\_Click({

UpdateDelegation

})

$FORM.FindName('RemoveToolkit').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Running toolkit removal script..."

If (Get-DscConfiguration | ? Name -eq ($FORM.FindName('ToolkitNameTextBox').Text + "EP") | ? Ensure -eq "Present")

{

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Toolkit endpoint was found as both present and active on this local machine, we can proceed with the removal."

$CommandToExecute = $FORM.FindName('ScriptOutputTextBlock').Text

$CommandToExecute = $CommandToExecute.Replace("#CleanAll", "CleanAll")

$CommandToExecute = $CommandToExecute.Replace("#" + $FORM.FindName('ToolkitNameTextBox').Text, $FORM.FindName('ToolkitNameTextBox').Text)

$CommandToExecute = $CommandToExecute.Replace("#Start-DscConfiguration", "Start-DscConfiguration")

If ((-not ($FORM.FindName('FilterModuleComboBox').Items -contains "Microsoft.SystemCenter.ServiceManagementAutomation")) -and ($FORM.FindName('BypassSMACmdletsNotPresentCB').IsChecked -eq $true))

{

#write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : SMA module was not found on the local machine. Removal script will be updated before actual removal."

$CommandToExecute = $CommandToExecute -replace ",Start-SmaRunbook,(.+?),(.+?),(.+?)", ""

$CommandToExecute = $CommandToExecute -replace ",Start-SmaRunbook,(.+?),,", ""

#$CommandToExecute = $CommandToExecute -replace ",Stop-SmaRunbook,(.+?),(.+?),(.+?)", ""

}

#write-host ($CommandToExecute)

write-host (Invoke-Expression ($CommandToExecute) -Verbose)

}

else

{

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : This toolkit was not found as both present and active on this local machine. We'll not be trying to remove it."

}

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Running toolkit removal script...done!"

})

$FORM.FindName('ImportToolkit').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Running toolkit import script..."

$CommandToExecute = $FORM.FindName('ScriptOutputTextBlock').Text

$CommandToExecute = $CommandToExecute.Replace("#" + $FORM.FindName('ToolkitNameTextBox').Text, $FORM.FindName('ToolkitNameTextBox').Text)

$CommandToExecute = $CommandToExecute.Replace("#Start-DscConfiguration", "Start-DscConfiguration")

If (-not ($FORM.FindName('FilterModuleComboBox').Items -contains "Microsoft.SystemCenter.ServiceManagementAutomation"))

{

If ($FORM.FindName('BypassSMACmdletsNotPresentCB').IsChecked -eq $true)

{

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : SMA module was not found on the local machine. Test toolkit will still be imported, but SMA cmdlets will be disabled."

$CommandToExecute = $CommandToExecute -replace ",Start-SmaRunbook,(.+?),(.+?),(.+?)", ""

$CommandToExecute = $CommandToExecute -replace ",Start-SmaRunbook,(.+?),,", ""

#$CommandToExecute = $CommandToExecute -replace ",Stop-SmaRunbook,(.+?),(.+?),(.+?)", ""

write-host (Invoke-Expression ($CommandToExecute) -Verbose)

popup -Message ("Toolkit was imported, please wait about 30 seconds before trying a test, while WinRM restarts")

}

else

{

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : SMA module was not found on the local machine. Test toolkit will not be imported, as the bypass checkbox is not enabled in the tool."

}

}

else

{

write-host (Invoke-Expression ($CommandToExecute) -Verbose)

popup -Message ("Toolkit was imported, please wait about 30 seconds before trying a test, while WinRM restarts")

}

#write-host $CommandToExecute

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Running toolkit import script...done!"

})

$FORM.FindName('TestToolkit').Add\_Click({

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Testing toolkit..."

If (Get-DscConfiguration | ? Name -eq ($FORM.FindName('ToolkitNameTextBox').Text + "EP") | ? Ensure -eq "Present")

{

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Toolkit endpoint was found as both present and active on this local machine, we can proceed with the test."

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Connecting to JEA session (with current logged on user)...This may take a few seconds..."

write-host (Invoke-Expression ("`$s = New-PSSession -cn . -ConfigurationName " + $FORM.FindName('ToolkitNameTextBox').Text +"EP"))

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Displaying available cmdlets..."

write-host (Invoke-Expression ("Invoke-command `$s {get-command} | out-string"))

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] Exiting JEA session..."

write-host (Invoke-Expression ("Remove-PSSession `$s"))

}

else

{

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : This toolkit was not found as both present and active on this local machine. Please make sure it is already imported, either manually or via the tool."

}

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Testing toolkit...done!"

})

########################################################################################

#Make sure we run elevated, or relaunch as admin

########################################################################################

$CurrentScriptDirectory = $PSCommandPath.Substring(0,$PSCommandPath.LastIndexOf("\"))

Set-Location $CurrentScriptDirectory

#Thanks to http://gallery.technet.microsoft.com/scriptcenter/63fd1c0d-da57-4fb4-9645-ea52fc4f1dfb

$IsAdmin = ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole] "Administrator")

if (-not $IsAdmin)

{

try

{

$ScriptToLaunch = (Get-Location -PSProvider FileSystem).Path + "\JEAToolkitHelper.ps1"

$arg = "-file `"$($ScriptToLaunch)`""

write-host -ForegroundColor yellow "["(date -format "HH:mm:ss")"] WARNING : This script should run with administrative rights - Relaunching the script in elevated mode in 3 seconds..."

start-sleep 3

Start-Process "$psHome\powershell.exe" -Verb Runas -ArgumentList $arg -ErrorAction 'stop'

}

catch

{

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] Error : Failed to restart script with administrative rights - please make sure this script is launched elevated."

break

}

exit

}

else

{

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"] We are running in elevated mode, we can proceed with launching the tool."

}

################################################

# Main

################################################

write-host -ForegroundColor green "["(date -format "HH:mm:ss")"] JEA Toolkit Helper v$ToolVersion"

$Global:SDDLAccountList = @()

$Global:strSDDL = $Global:DefaultSDDL

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] Checking parameters and prerequisites..."

$Global:SMAIntegration = $false

$Global:SMAWS = $SMAEndPointWS

$Global:SMAPort = $SMAEndPointPort

If ($SMAWS)

{

$Global:FullRunbookList = @()

$Global:FullRunbookList = Invoke-Command -ScriptBlock {

param ($WS,$Port)

get-smarunbook -WebServiceEndpoint $WS -Port $Port | select RunbookName, RunbookID, Tags, Description

$runbooks

} -ArgumentList $Global:SMAWS, $Global:SMAPort -ComputerName ($Global:SMAWS).split("//")[2]

#write-host $Global:FullRunbookList

If (($Global:FullRunbookList.Count -eq 0) -or ($Global:FullRunbookList -eq $mull) -or ($Global:FullRunbookList -eq ""))

{

write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : Could not connect to SMA server or no runbooks were found. SMA integration will be disabled this time"

$Global:SMAIntegration = $false

}

else

{

$PropertyArray = @()

Foreach ($Runbook in $Global:FullRunbookList)

{

$tmpObject2 = select-object -inputobject "" RunbookChecked, RunbookName

$tmpObject2.RunbookChecked = $false

$tmpObject2.RunbookName = $Runbook.RunbookName

$PropertyArray += $tmpObject2

$FORM.FindName('PickRunbooksComboBox').ItemsSource = $PropertyArray

}

write-host -ForegroundColor gray "["(date -format "HH:mm:ss")"]" $Global:FullRunbookList.Count "Runbooks were retrieved for SMA integration."

$Global:SMAIntegration = $true

}

}

else

{

write-host -ForegroundColor white "["(date -format "HH:mm:ss")"] No SMA server specified. SMA integration will not be available this time."

$Global:SMAIntegration = $false

}

If ($Global:SMAIntegration -eq $false)

{

$FORM.FindName('PickRunbookLabelContainer').IsEnabled = $false

$FORM.FindName('PickRunbooksComboBox').IsEnabled = $false

$FORM.FindName('AddRunbookToGrid').IsEnabled = $false

}

$Global:PropertyArray = New-Object System.Collections.ArrayList

$Global:CmdletList = Get-Command | Sort-Object Name | Select Name

UpdateCmdletList

UpdateModuleList

If (-not ($FORM.FindName('FilterModuleComboBox').Items -contains "Microsoft.SystemCenter.ServiceManagementAutomation"))

{write-host -ForegroundColor red "["(date -format "HH:mm:ss")"] WARNING : SMA module was not found on the local machine. Test toolkit import may only be limited to non-SMA cmdlets."}

$FORM.FindName('ImportCSVFileAction').Items.Add("Replace grid content") | out-null

$FORM.FindName('ImportCSVFileAction').Items.Add("Add to grid content") | out-null

$FORM.FindName('ImportCSVFileAction').Text = "Replace grid content"

$Global:CommandArray = New-Object System.Collections.ArrayList

$Global:CommandArray = @()

write-host -ForegroundColor green "["(date -format "HH:mm:ss")"] Displaying GUI..."

$FORM.ShowDialog() | Out-Null

write-host -ForegroundColor green "["(date -format "HH:mm:ss")"] Exiting GUI..."