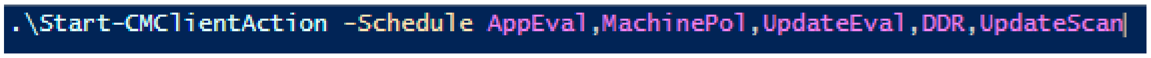
Script to run few Actions from Control Panel – Configuration Manager

In this guide, I will show you how to use the script to run few actions from control panel – configuration manager after IPU or Bare Metal or OS Refresh. I was able to get the script from SCCM Community MVP where, I posted the request and it was answered. Thought it might help others as well.

At the current project, I run the script as the last step before leaving the office so w/s can get all the policy applied and any updates or applications to be installed will be completed by early morning so users don't have to wait for an update or application to install. This reduce their wait time to start their daily routine.

First thing you will have to copy the file to a network share from where you can access the script. The procedure to run the script are as follows:

1. Copy the PowerShell Script from network share to local C:\Temp or any folder of your choice
2. Type PowerShell – Right Click - Open File Location – Right Click Run as Admin or DSS Support
3. Go to the path where you have the script copied – In my case C:\Temp
4. Run this command - **Start-CMClientAction -Schedule AppEval,MachinePol,UpdateEval,DDR,UpdateScan**
5. 

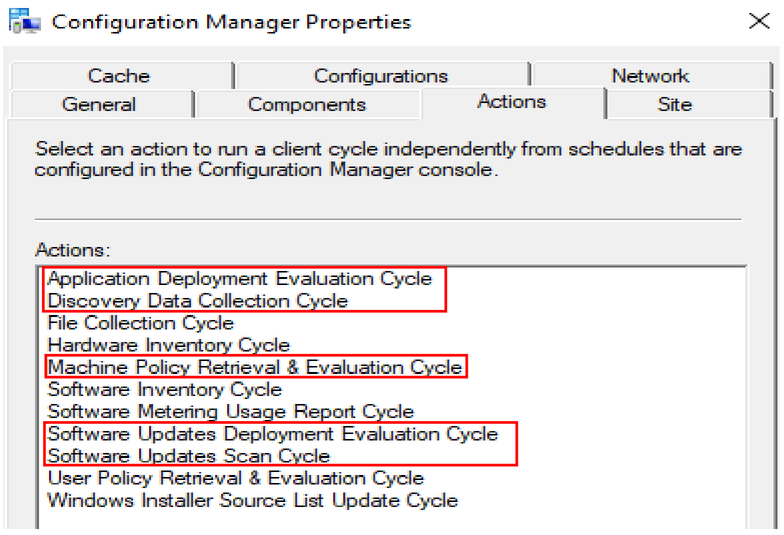
```
.\Start-CMClientAction -Schedule AppEval,MachinePol,UpdateEval,DDR,UpdateScan
```

6. If there is no error it will return to command prompt – if there is any error then you will see red message. In my case the command executed successfully

Now the workstation will reach out to the server and refresh above policy and any updates or application that needs to be installed will be carried out automatically within Software Center.

I choose only few of the actions that are vital for the project. You can run other actions, if you need so.

Here is the screen shot of the actions that, I choose to run.



Configuration Manager Properties

Select an action to run a client cycle independently from schedules that are configured in the Configuration Manager console.

Actions:

Application Deployment Evaluation Cycle
Discovery Data Collection Cycle
File Collection Cycle
Hardware Inventory Cycle
Machine Policy Retrieval & Evaluation Cycle
Software Inventory Cycle
Software Metering Usage Report Cycle
Software Updates Deployment Evaluation Cycle
Software Updates Scan Cycle
User Policy Retrieval & Evaluation Cycle
Windows Installer Source List Update Cycle

Thanks

Ram Lan
12th Jul 2019

Note – The Script file name is Start-CMClientAction.ps1

```
function Start-CMClientAction {
    <#
.SYNOPSIS
    Invokes CM Client actions on local or remote machines

.DESCRIPTION
    This script will allow you to invoke a set of CM Client actions on a machine (with
optional credentials), providing a list of the actions and an optional delay betweens
actions.
    The function will attempt for a default of 5 minutes to invoke the action, with a
10 second delay inbetween attempts. This is to account for invoke-wmimethod failures.

.PARAMETER Schedule
       Define the schedules to run on the machine - 'HardwareInv', 'FullHardwareInv',
'SoftwareInv', 'UpdateScan', 'UpdateEval', 'MachinePol', 'AppEval', 'DDR'

.PARAMETER Delay
       Specify the delay in seconds between each schedule when more than one is ran -
0-30 seconds

.PARAMETER ComputerName
    Specifies the computers to run this against

.PARAMETER Timeout
    Specifies the timeout in minutes after which any individual computer will stop
attempting the schedules. Default is 5 minutes.

.PARAMETER Credential
       Optional PSCredential

.EXAMPLE
       # Start a machine policy eval and a hardware inventory cycle
       Start-CMClientAction -Schedule MachinePol,HardwareInv or
    Start-CMClientAction -Schedule AppEval,MachinePol,UpdateEval,DDR,UpdateScan

.NOTES
    FileName:    Start-CMClientAction.ps1
    Author:      Cody Mathis
    Contact:     @CodyMathis123
    Created:     11-29-2018
    Updated:     03-09-2019
#>
    [CmdletBinding(SupportsShouldProcess = $true)]
    param
    (
        [parameter(Mandatory = $false, ValueFromPipelineByPropertyName = $true)]
        [Alias('Computer', 'HostName', 'ServerName', 'IPAddress')]
        [string[]]$ComputerName = $env:COMPUTERNAME,
        [parameter(Mandatory = $true)]
        [ValidateSet('HardwareInv', 'FullHardwareInv', 'SoftwareInv', 'UpdateScan',
'UpdateEval', 'MachinePol', 'AppEval', 'DDR')]
        [string[]]$Schedule,
        [parameter(Mandatory = $false)]
        [ValidateRange(0, 30)]
        [int]$Delay = 5,
        [parameter(Mandatory = $false)]
        [int]$Timeout = 5,
        [parameter(Mandatory = $false)]
        [pscredential]$Credential
    )
    begin {
        $TimeSpan = New-TimeSpan -Minutes $Timeout
    }
    process {
        foreach ($Computer in $ComputerName) {
            foreach ($Option in $Schedule) {
                $Action = switch -Regex ($Option) {
                    '^HardwareInv$|^FullHardwareInv$' {
                        '{00000000-0000-0000-0000-000000000001}'
                    }
                    '^SoftwareInv$' {
                        '{00000000-0000-0000-0000-000000000002}'
                    }
                    '^UpdateScan$' {
                        '{00000000-0000-0000-0000-000000000113}'
                    }
                    '^UpdateEval$' {
                        '{00000000-0000-0000-0000-000000000108}'
                    }
                    '^MachinePol$' {
                        '{00000000-0000-0000-0000-000000000021}'
                    }
                    '^AppEval$' {
                        '{00000000-0000-0000-0000-000000000121}'
                    }
                    '^DDR$' {
                        '{00000000-0000-0000-0000-000000000003}'
                    }
                }

                $StopWatch = [System.Diagnostics.Stopwatch]::StartNew()
                do {
                    try {
                        Remove-Variable MustExit -ErrorAction SilentlyContinue
                        Remove-Variable Invocation -ErrorAction SilentlyContinue
                        if ($Option -eq 'FullHardwareInv') {
                            $getWMIObjectSplat = @{
                                ComputerName = $Computer
                                Namespace    = 'root\ccm\invagt'
                                Class        = 'InventoryActionStatus'
                                Filter       = "InventoryActionID ='$Action'"
                                ErrorAction  = 'Stop'
                            }
                            if ($PSBoundParameters.ContainsKey('Credential')) {
                                $getWMIObjectSplat.Add('Credential', $Credential)
                            }
                            Write-Verbose "Attempting to delete Hardware Inventory
history for $Computer as a FullHardwareInv was requested"
                            $HWInv = Get-WMIObject @getWMIObjectSplat
                            if ($null -ne $HWInv) {
                                $HWInv.Delete()
                                Write-Verbose "Hardware Inventory history deleted for
$Computer"
                            }
                            else {
                                Write-Verbose "No Hardware Inventory history to delete
for $Computer"
                            }
                        }
                        $invokeWmiMethodSplat = @{
                            ComputerName = $Computer
                            Name         = 'TriggerSchedule'
                            Namespace    = 'root\ccm'
                            Class        = 'sms_client'
                            ArgumentList = $Action
                            ErrorAction  = 'Stop'
                        }
                        if ($PSBoundParameters.ContainsKey('Credential')) {
                            $invokeWmiMethodSplat.Add('Credential', $Credential)
                        }
                        Write-Verbose "Triggering a $Option Cycle on $Computer via the
'TriggerSchedule' WMI method"
                        $Invocation = Invoke-WmiMethod @invokeWmiMethodSplat
                    }
                    catch [System.UnauthorizedAccessException] {
                        Write-Error -Message "Access denied to $Computer" -Category
AuthenticationError -Exception $_.Exception
                        $MustExit = $true
                    }
                    catch {
                        Write-Warning "Failed to invoke the $Option cycle via WMI.
Will retry every 10 seconds until [StopWatch $($StopWatch.Elapsed) -ge $Timeout
minutes] Error: $($_.Exception.Message)"
```

```
                        Start-Sleep -Seconds 10
                    }
                }
                until ($Invocation -or $StopWatch.Elapsed -ge $TimeSpan -or $MustExit)
                if ($Invocation) {
                    Write-Verbose "Successfully invoked the $Option Cycle on $Computer
via the 'TriggerSchedule' WMI method"
                    Start-Sleep -Seconds $Delay
                }
                elseif ($StopWatch.Elapsed -ge $TimeSpan) {
                    Write-Error "Failed to invoke $Option cycle via WMI after $Timeout
minutes of retrrying."
                }
                $StopWatch.Reset()
            }
        }
    }
    end {
        Write-Verbose "Following actions invoked - $Schedule"
    }
}
```