Start-DeploymentProUserBatch.ps1 Specification

Contents

[1. DOCUMENT OVERVIEW 2](#_Toc298935650)

[2. Pre-Process Setup 2](#_Toc298935651)

[3. DOCUMENT HISTORY 2](#_Toc298935652)

# Script Description

Start-DeploymentProUserBatch.ps1 is designed to allow scheduling of CSV batch of users under a single MSPC customer. The function has the ability to use a System.Datetime as input for a scheduled time/date or a now switch that will kick off the users on next check in.

This script takes a CSV input consisting of the PrimaryEmailAddress which is the value displayed in MSPC users list and the intended destination email address expressed a DestinationEmailAddress in the CSV which is manually populated by the executing user. The script will guide the user through scheduling users including CSV input validation, MSPC credential validation, and start date/time validation. All successes and failures will be logged to a log file location output in the console. Each run of the script is logged independently.

# Requirements

* Windows Operating System
* BitTitan Powershell SDK
* Microsoft Windows PowerShell 4.0+
* Microsoft .NET Framework 3.5
* Microsoft .NET Framework 4.6.2
* The ability to run Powershell as an administrator
* API access granted to BitTitan MSPC account executing the script.
* Powershell execution policy set to RemoteSigned or Unrestricted. If already set to Unrestricted do not change to RemoteSigned!
* Ability to create folder on the root of C:\

# Code explanation

This is a simple logging function that allows a text file to be written with log messages pertaining to the code process flow.

function \_Log

{

param ( $Message )

"[$(Get-Date -format 'G') | $($pid) | $($env:username)] $Message" | Out-File -FilePath $Logfile -Append

}

This function will attempt to create a working directory for the log files and statistics CSV files to be stored in if ones does not exist.

function New-StorageDirectory

{

$Directory = "C:\Migrations\_BitTitan"

if ( ! (Test-Path $Directory))

{

try

{

New-Item -ItemType Directory -Path $Directory -Force -ErrorAction Stop

}

catch

{

\_Log -Message "Failed to create working directory at - $Directory."

$Directory = "$HOME\Desktop\Migrations\_BitTitan"

New-Item -ItemType Directory -Path $Directory -Force

}

if ( $Directory )

{

$Directory

}

}

else

{

Get-Item -Path $Directory | select FullName

}

}

This function will attempt to utilize the Select-CSVDialog function. The loop will allow five attempts at locating the script file before exiting.

function Import-CSVBatchFile

{

\_Log -Message "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Import-CSVBatchFile\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

$i = 0

do

{

Write-Output "Please select your CSV from the pop out dialog box."

\_Log -Message "Please select your CSV from the pop out dialog box."

Start-Sleep -Seconds 3

$csv = Select-CSVDialog

$i++

\_Log -Message "File $($csv) was selected, continuing..."

if($csv -like '\*.csv')

{

Write-Output "CSV has been found, proceeding..."

\_Log -Message "CSV provided by input was correct, proceeding..."

$script:users = @(Import-Csv -Path $csv)

\_Log -Message "$($script:users.count) items found in CSV..."

}

else

{

Write-Error "The CSV file location is not valid, please try again..."

\_Log -Message "CSV provided was NOT correct, please try again..."

$csv = $null

}

}

until($i -ge 5 -or ($csv -ne $null))

if($i -ge 5 -and $csv -eq $null)

{

Write-Error "This script cannot continue due to no valid CSV being selected, please run the script again!"

\_Log -Message "Script loop was not satisified within 5 attempts, script has been exited"

Start-Sleep -Seconds 10

Break

}

}

This function provides a method for a file selection GUI dialog to select the batch file CSV.

function Select-CSVDialog

{

param([string]$Title="Please select the batch file CSV",[string]$Directory=(Get-Location).Path,[string]$Filter="Comma Seperated Values | \*.csv")

[System.Reflection.Assembly]::LoadWithPartialName("PresentationFramework") | Out-Null

$objForm = New-Object Microsoft.Win32.OpenFileDialog

$objForm.InitialDirectory = $Directory

$objForm.Filter = $Filter

$objForm.Title = $Title

$Show = $objForm.ShowDialog()

If ($Show -eq "OK")

{

Return $objForm.FileName

}

Else

{

Write-Error "Operation cancelled by user."

}

}

Attempts to import the BitTitanPowerShell module if it isn't already loaded under the shell context.

function New-BitTitanPSSession

{

\_Log -Message "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*New-BitTitanPSSession\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

$module = Get-Module -Name "BitTitanPowerShell" -ErrorAction SilentlyContinue

if(-not $module)

{

try

{

Import-Module "C:\Program Files (x86)\BitTitan\BitTitan PowerShell\BitTitanPowerShell.dll" -ErrorAction Stop

\_Log -Message "Successfully added BitTitanPowerShell module, proceeding!"

}

catch

{

\_Log -Message "Could not add BitTitanPowerShell module! The please ensure the updated BitTitan Powershell SDK is up to date and installed!"

throw

}

}

else

{

\_Log -Message "BitTitanPowerShell module is already loaded, skipping add process!"

}

}

Will remove the BitTitanPowerShell module if it's currently loaded into the PSSession, if the module does not exist in the current session the logic will be skipped and the state logged.

function Remove-BitTitanPSSession

{

\_Log -Message "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Remove-BitTitanPSSession\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

$module = Get-Module -Name "BitTitanPowerShell" -ErrorAction SilentlyContinue

if($module)

{

try

{

Remove-Module -Name "BitTitanPowerShell" -ErrorAction Stop

\_Log -Message "Successfully removed BitTitanPowerShell module!"

}

catch

{

\_Log -Message "Could not remove BitTitanPowerShell module due to`r`n$($\_.Exception)!"

throw

}

}

else

{

\_Log -Message "BitTitanPowerShell module is not currently loaded, skipping!"

}

}

This function will attempt to take a CSV input of PrimaryEmailAddress and DestinationEmailAddress and attempt to schedule each user in the CSV against a single MSPC customer. The scheduling time is based on two separate parameter sets, one including a now switch which will pull the latest UTC time and pass it into the module and the other taking the script user DateTime input and passing it in with correct translation.

function Start-DeploymentProUserBatch

{

param

(

[CmdletBinding()]

[Parameter(Mandatory=$true)]

[array]$Batch,

[Parameter(Mandatory=$true,ParameterSetName='StartDate')]

[DateTime]$StartDate,

[Parameter(Mandatory=$true,ParameterSetName='Now')]

[switch]$Now

)

\_Log -Message "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Start-DeploymentProUserBatch\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

New-BitTitanPSSession

#The following do/until block will attempt to authenticate to MSPC and impersonate the customerId provided through the console input. The user will be allowed five attempts at successfully authenticating to MSPC.

$k = 0

do

{

$k++

$ticket = $null

$ticketwithoutorganization = $null

$customer = $null

Write-Output "Please Enter Your BitTitan MSPC/MigrationWiz Credentials"

Start-Sleep -Seconds 3

$cred = Get-Credential -Message "Enter Your MSPC/MigrationWiz Credential:"

[guid]$customerid = Read-Host "Please provide the MSPC Customer ID"

$ticketwithoutorganization = Get-BT\_Ticket -Credentials $Cred -ServiceType BitTitan -Environment "BT"

if($ticketwithoutorganization -and $customerid)

{

Write-Output "MSPC/MigrationWiz credentials are valid, attempting to gather customer information..."

\_Log -Message "MigrationWiz credentials provided were correct, proceeding to attempt to gather customer information..."

$customer = Get-BT\_Customer -Ticket $ticketwithoutorganization -Environment "BT" -FilterBy\_Guid\_Id $customerId.Guid

if($customer)

{

try

{

$Ticket = Get-BT\_Ticket -Credentials $cred -ServiceType BitTitan -Environment "BT" -OrganizationId $customer.OrganizationId -ImpersonateId $customer.SystemUserId -ErrorAction Stop

Write-Output "Ticket was set successfully, proceeding..."

\_Log -Message "Ticket was set successfully, proceeding..."

}

catch

{

Write-Error "Ticket could not be set on this attempt!"

\_Log -Message "Ticket could not be set on this attempt due to $($\_.Exception), please try again!"

}

}

else

{

Write-Error "MSPC/MigrationWiz credentials provided valid but a customer could not identified by the customerID, please modify the function input and try again!"

\_Log -Message "MSPC/MigrationWiz credentials provided were valid but no customer could be identified by the customerID, please make sure the customerID provided is valid and try again..."

}

}

else

{

Write-Error "MSPC/Migrationwiz credentials provided were not valid to obtain a ticket, please try again!"

\_Log -Message "MSPC/MigrationWiz credentials provided could not obtain a ticket, please try again..."

}

}

until($k -ge 5 -or ($ticket -ne $null))

#This loop will exit the script if valid MigrationWiz credentials are not provided within 5 or more attempts.

if($k -ge 5 -or $ticket -eq $null)

{

Write-Error "This function cannot continue due to no valid credentials being provided for the MSPC/MigrationWiz service, please run the function again!"

\_Log -Message "MSPC/MigrationWiz credential loop was not satisified within 5 attempts, script has been exited"

Start-Sleep -Seconds 10

Break

}

#If the CSV contains the correct values the code will execute setting necessary variables and also running queries against MSPC to ensure that the user is eligible for DeploymentPro scheduling. If the user is eligible, they will be scheduled based on the parameter input to the function which is handled outside of the function.

if(($users.count -gt 0) -and ($users.PrimaryEmailAddress -gt 0) -and ($users.DestinationEmailAddress -gt 0))

{

foreach($user in $users)

{

$mspcuser = $null

$mspcUser += @(Get-BT\_CustomerEndUser -Ticket $Ticket -FilterBy\_String\_PrimaryEmailAddress $user.PrimaryEmailAddress -FilterBy\_Guid\_OrganizationId $ticket.OrganizationId -Environment "BT" -FilterBy\_Boolean\_IsDeleted $false)

if($mspcUser)

{

foreach($endUser in $mspcUser)

{

try

{

if($Now)

{

$dateTime = [DateTime]::UtcNow.ToString('o')

Start-BT\_DpUser -Ticket $ticket -UserIdentity $endUser.PrimaryEmailAddress -DestinationEmailAddress $user.DestinationEmailAddress -CustomerId $customerId.Guid -ProductSkuId $dplicense -StartTime $datetime -Environment BT -ErrorAction Stop

}

elseif($startDate -ne $null)

{

$dateTime = ($startDate.ToUniversalTime()).ToString('o')

Start-BT\_DpUser -Ticket $ticket -UserIdentity $endUser.PrimaryEmailAddress -DestinationEmailAddress $user.DestinationEmailAddress -CustomerId $customerId.Guid -ProductSkuId $dplicense -StartTime $datetime -Environment BT -ErrorAction Stop

}

\_Log -Message "User $($endUser.PrimaryEmailAddress) was successfully scheduled to $($dateTime)"

Write-Output "User $($endUser.PrimaryEmailAddress) was successfully scheduled to $($dateTime)"

$modules = Get-BT\_CustomerDeviceUserModule -Ticket $ticket -FilterBy\_Boolean\_IsDeleted $false -FilterBy\_Guid\_EndUserId $endUser.Id -FilterBy\_String\_ModuleName "outlookconfigurator" -FilterBy\_Guid\_OrganizationId $ticket.OrganizationId -Environment "BT"

if($modules)

{

foreach($module in $modules)

{

try

{

$machineName = Get-BT\_CustomerDevice -Ticket $ticket -FilterBy\_Guid\_Id $module.DeviceId

$moduleStartDate = Set-BT\_CustomerDeviceUserModule -Ticket $ticket -customerdeviceusermodule $module -ScheduledStartDate $dateTime -ErrorAction Stop

\_Log -Message "Successfully updated user $($endUser.PrimaryEmailAddress),machine $($machineName.devicename) scheduledstartdate module property to $($dateTime)"

Write-Output "Successfully updated user $($endUser.PrimaryEmailAddress),machine $($machineName.devicename) scheduledstartdate module property to $($dateTime)"

}

catch

{

\_Log -Message "Could not update machine $($machineName.devicename) scheduledstartdate property for MSPC user $($endUser.PrimaryEmailAddress) due to the following exception`r`n$($\_.Exception.Message)"

Write-Error "Could not update machine $($machineName.devicename) scheduledstartdate property for MSPC user $($endUser.PrimaryEmailAddress) due to the following exception`r`n$($\_.Exception.Message)"

}

}

}

else

{

\_Log -Message "$($endUser.PrimaryEmailAddress) was found in the MSPC customer but does NOT have any DeploymentPro modules!"

Write-Warning "$($endUser.PrimaryEmailAddress) was found in the MSPC customer but does NOT have any DeploymentPro modules!"

}

}

catch

{

\_Log -Message "User $($endUser.PrimaryEmailAddress) was not scheduled due to the following exception`r`n$($\_.Exception.Message)"

Write-Error "User $($endUser.PrimaryEmailAddress) was not scheduled due to the following exception`r`n$($\_.Exception.Message)"

}

}

}

else

{

if(-not $mspcUser)

{

Write-Warning "User $($user.PrimaryEmailAddress) was not found in this MSPC organization, skipping!"

\_Log -Message "User $($user.PrimaryEmailAddress) was not found in this MSPC organization, skipping!"

}

}

}

}

else

{

Write-Error "No users were found in the CSV or the CSV supplied was invalid, please check the CSV and run the function again"

\_Log -Message "Start-DeploymentProUserBatch failed due to no users being found in the supplied CSV value, function was aborted"

}

Remove-BitTitanPSSession

}

Block will ensure that the storage directory is created as well as setting a log file variable to be used for the session. The block will execute the CSV import function to be used by the rest of the script.

$StorageDirectory = New-StorageDirectory

[string]$Logfile = $StorageDirectory.FullName + "\StartDeploymentProUserBatch" + (Get-Date -Format "MMddyyTHHmmss") + ".log"

Write-Output "`r`nPlease refer to the log file for additional information located in the following location!`r`n`n$($logfile)"

Start-Sleep -Seconds 2

$users = $null

Write-Output "Please provide a CSV that contains the PrimaryEmailAddress and DestinationEmailAddress values!"

Start-Sleep -Seconds 2

Import-CSVBatchFile

$dplicense = "6D8A5E88-2116-497B-874F-38663EF0EBE8"

If the CSV contains valid headers and the prior yes/no choice prompt returns a yes value then the following block is entered. An additional Yes/No prompt will be given to ask the user to specify a date/time to execute the DeploymentPro switch. If the user selects no then the current date/time will be used. If the script user indicates yes then first if block will be entered and another prompt will be given to ensure that the date/time entered by the script user is correct/desired. If the user selects no the script breaks as no valid value was entered.

if(($users | Get-Member DestinationEmailAddress) -and ($users | Get-Member PrimaryEmailAddress))

{

\_Log -Message "The CSV provided is valid for Start-DeploymentProUserBatch, proceeding..."

Write-Output "The CSV provided is valid for Start-DeploymentProUserBatch, proceeding..."

$title = "DeploymentPro Switch Date/Time?"

$message = "Do you want to specify a date/time value for the user batch to be executed?"

$yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes", `

"You will be prompted to enter a valid System.DateTime value."

$no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", `

"The UTC Time/Date value $([DateTime]::UtcNow.AddHours(-24).ToString('o')) will be used."

$options = [System.Management.Automation.Host.ChoiceDescription[]]($yes, $no)

$result = $host.ui.PromptForChoice($title, $message, $options, 1)

if($result -eq '0')

{

[DateTime]$inputdate = Read-Host "Please provide a valid System.DateTime value"

if($inputdate -ne $null)

{

$title = "DeploymentPro Switch Date/Time Correct?"

$message = "$($inputdate) will be used, is this correct?"

$yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes", `

"The value displayed is correct and will be used by all objects in the batch"

$no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", `

"The value displayed is NOT correct and the script will be aborted"

$options = [System.Management.Automation.Host.ChoiceDescription[]]($yes, $no)

$result = $host.ui.PromptForChoice($title, $message, $options, 1)

if($result -eq '0')

{

Write-Output "Executing Start-DeploymentProUserBatch with $($inputdate) execution time..."

Start-DeploymentProUserBatch -Batch $users -StartDate $inputdate.ToString('o')

}

else

{

Write-Error "The input for the DeploymentPro Switch Date/Time was incorrect, aborting script execution. Please run the script again!"

\_Log -Message "The input for the DeploymentPro Switch Date/Time was incorrect, aborting script execution. Please run the script again!"

Break

}

}

else

{

Write-Error "A valid System.DateTime input was not provided, the script has been aborted. Please run the script again!"

\_Log -Message "A valid System.DateTime input was not provided, the script has been aborted. Please run the script again!"

Break

}

}

if($result -eq '1')

{

Write-Output "Executing Start-DeploymentProUserBatch with $([DateTime]::Now.ToString()) execution time..."

Start-DeploymentProUserBatch -Batch $users -Now

}

if($result -eq '-1')

{

Write-Error "No input was provided to the DeploymentPro Switch Date/Time dialog box, the script cannot continue please restart the script!"

\_Log -Message "No input was provided to the DeploymentPro Switch Date/Time dialog box, the script cannot continue please restart the script!"

Break

}

}

else

{

\_Log -Message "The CSV supplied did not contain valid values, please run script again and provide a CSV containing the PrimaryEmailAddress, DestinationEmailAddress!"

Write-Error "The CSV supplied did not contain valid values, please run the script again and provide a CSV containing the PrimaryEmailAddress, DestinationEmailAddress!"

}

# Usage

1. Download Start-DeploymentProUserBatch.ps1 and place into C:\Scripts, if the folder does not exist please create it.
2. Open Windows Powershell as an administrator
3. Change directory to the C:\Scripts directory.
4. Run the following to execute the script.
5. .\Start-DeploymentProUserBatch.ps1

# DOCUMENT HISTORY

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision | Changes | Author |
| 01/23/2017 | 1 | Initial Draft | CJ |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |