Cherwell GLOBAL CONFERENCE 2016

Innovation with Purpose.

Share

Building Powerful Workflow Automation with Cherwell and PowerShell

Robert Goguen
Excalibur Data Systems
President Canadian Operations

Jeff Jones
Excalibur Data Systems, Inc.
Consultant

Richie Ritter
Western Carolina University
Systems Engineer

Agenda

- Welcome & Session Introduction
- What is PowerShell?
  - PowerShell ISE
  - Commands/Cmd-Lets
  - Operators
  - Variables
  - Flow Control
- LAB 1 – Exploring PowerShell
- PowerShell Scripting
  - Installing the Latest Version
  - Execution Policies
  - Script Files
  - Parameters
  - Output
  - Modules

Agenda

- LAB 2 – PowerShell Scripts
- Using Scripting in Cherwell
  - Using a One-Step
  - Calling the script
    - Setting the Parameters
    - Getting the Output
    - Handling Errors
  - Stored Script or On The Fly
- LAB 3 – Calling PowerShell Scripts from Cherwell One-Steps
- Cherwell Orchestration Packs
  - Active Directory
  - Exchange
  - VMWare vRealize
- Q & A

Building Powerful Workflow Automation with Cherwell and PowerShell

Your Mission…..

- Increase Value to the Business
- Reduce Cost
- Minimize Risk
- Improve Service Offerings
- No impact on Quality or Services

What is PowerShell?

- What is PowerShell?

PowerShell is an automation platform and scripting language for Windows and Windows Server that allows you to simplify the management of your systems. Unlike other text-based shells, PowerShell harnesses the power of the .NET Framework, providing rich objects and a massive set of built-in functionality for taking control of your Windows environments (Source: MSDN)

- Why PowerShell?

Most Microsoft Systems are managed using PowerShell. It is Microsoft's go-forward technology that will be used to manage all products in the future.

What is PowerShell?

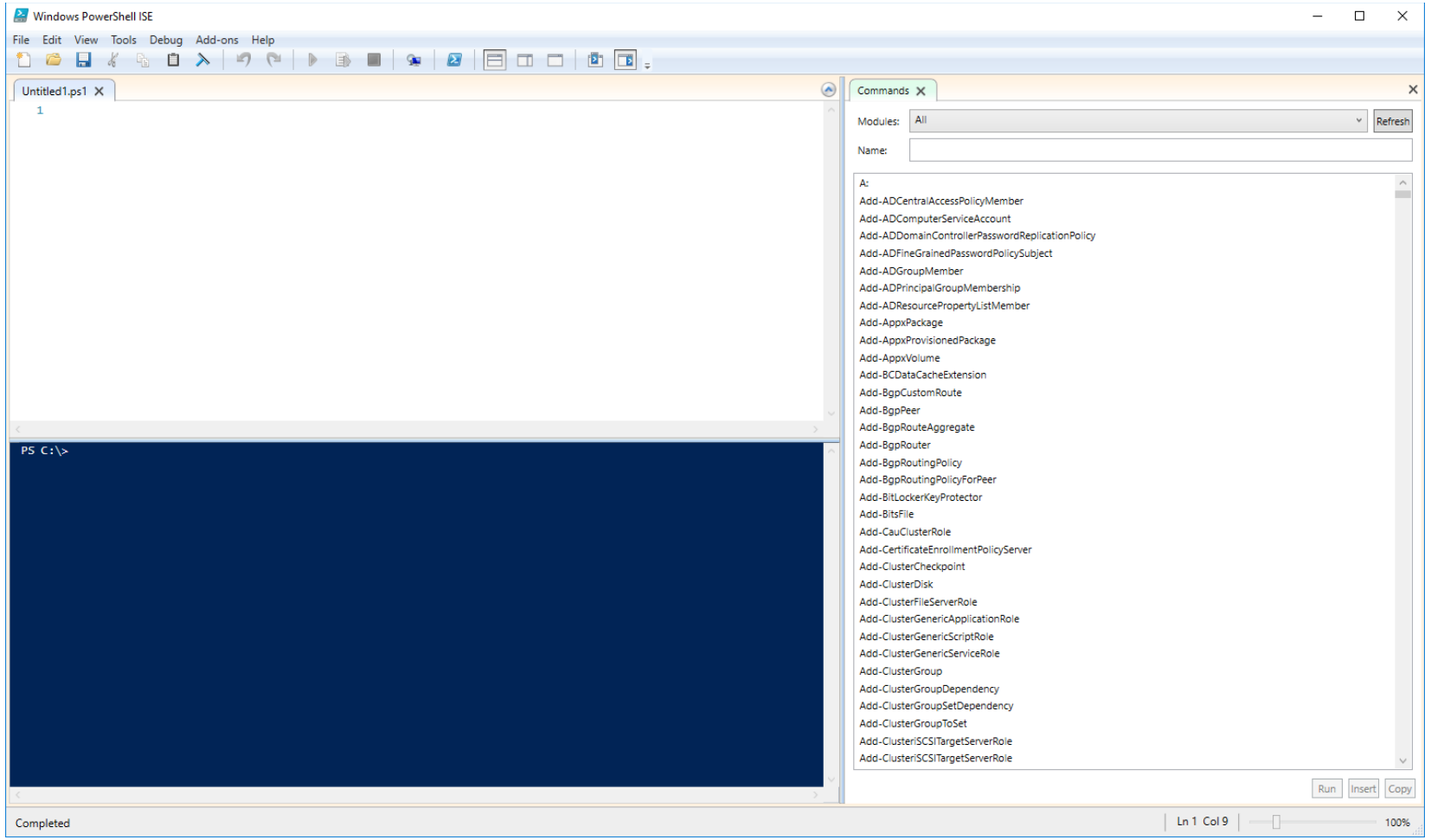
```
Cmdlet          Set-PSBreakpoint                    Microsoft.PowerShell.Utility
Cmdlet          Set-PSDebug                         Microsoft.PowerShell.Core
Cmdlet          Set-PSSessionConfiguration          Microsoft.PowerShell.Core
Cmdlet          Set-ScheduledJob                    PSScheduledJob
Cmdlet          Set-ScheduledJobOption              PSScheduledJob
Cmdlet          Set-Service                         Microsoft.PowerShell.Management
Cmdlet          Set-StrictMode                      Microsoft.PowerShell.Core
Cmdlet          Set-TraceSource                     Microsoft.PowerShell.Utility
Cmdlet          Set-Variable                        Microsoft.PowerShell.Utility
Cmdlet          Set-WmiInstance                     Microsoft.PowerShell.Management
Cmdlet          Set-WSManInstance                   Microsoft.WSMan.Management
Cmdlet          Set-WSManQuickConfig                Microsoft.WSMan.Management
Cmdlet          Show-Command                        Microsoft.PowerShell.Utility
Cmdlet          Show-ControlPanelItem               Microsoft.PowerShell.Management
Cmdlet          Show-EventLog                       Microsoft.PowerShell.Management
Cmdlet          Sort-Object                         Microsoft.PowerShell.Utility
Cmdlet          Split-Path                          Microsoft.PowerShell.Management
Cmdlet          Start-BitsTransfer                  BitsTransfer
Cmdlet          Start-Job                           Microsoft.PowerShell.Core
Cmdlet          Start-Process                       Microsoft.PowerShell.Management
Cmdlet          Start-Service                       Microsoft.PowerShell.Management
Cmdlet          Start-Sleep                         Microsoft.PowerShell.Utility
Cmdlet          Start-Transaction                   Microsoft.PowerShell.Management
Cmdlet          Start-Transcript                    Microsoft.PowerShell.Host
Cmdlet          Stop-Computer                       Microsoft.PowerShell.Management
Cmdlet          Stop-Job                            Microsoft.PowerShell.Core
Cmdlet          Stop-Process                        Microsoft.PowerShell.Management
Cmdlet          Stop-Service                        Microsoft.PowerShell.Management
Cmdlet          Stop-Transcript                     Microsoft.PowerShell.Host
Cmdlet          Suspend-BitsTransfer                BitsTransfer
Cmdlet          Suspend-Job                         Microsoft.PowerShell.Core
Cmdlet          Suspend-Service                     Microsoft.PowerShell.Management
Cmdlet          Tee-Object                          Microsoft.PowerShell.Utility
Cmdlet          Test-AppLockerPolicy                AppLocker
Cmdlet          Test-ComputerSecureChannel          Microsoft.PowerShell.Management
Cmdlet          Test-Connection                     Microsoft.PowerShell.Management
Cmdlet          Test-ModuleManifest                 Microsoft.PowerShell.Core
Cmdlet          Test-Path                           Microsoft.PowerShell.Management
Cmdlet          Test-PSSessionConfigurationFile     Microsoft.PowerShell.Core
Cmdlet          Test-WSMan                          Microsoft.WSMan.Management
```

What can I do with PowerShell?

Just About Anything!

100’s of Cmdlet’s, Functions and 1000’s of Parameters

PowerShell ISE



Commands/Cmd-Lets

- Cmdlets are the bones of PowerShell.
- Follow a naming pattern of Verb-Noun where most verbs are standardized and most nouns have multiple verbs.
- Some Examples:
  - Get-Help
  - Get-Date, Set-Date
  - Get-Service, New-Service, Start-Service, Stop-Service, Restart-Service
- To see cmdlets for a given ***verb*** you can use PS C:\> Get-Command -Noun ***verb***
  - *i.e.* - PS C:\> Get-Command -Noun service

Commands/Cmd-Lets

- Important or useful cmdlets
  - Get Help
    - PS C:\> Get-Help *Some-Cmdlet* -Show help for Some-Cmdlet
    - PS C:\> Get-Help *Some-Cmdlet* -full -Show detailed help
    - PS C:\> Get-Help *Some-Cmdlet* -online -Open the web page for the cmdlet
    - PS C:\> Get-Help *Some-Cmdlet* -examples -Show cmdlet usage examples
    - PS C:\> Get-Help **findsomething** -Search the help system for commands

Comparison Operators

- -eq Equal
- -ne Not equal
- -ge Greater than or equal
- -gt Greater than
- -lt Less than
- -le Less than or equal
- -like Wildcard comparison
- -notlike Wildcard comparison
- -match Regular expression comparison
- -notmatch Regular expression comparison
- -replace Replace operator
- -contains Containment operator
- -notcontains Containment operator
- -shl Shift bits left (PowerShell 3.0)
- -shr Shift bits right – preserves sign for signed values.(PowerShell 3.0)
- -in Like –contains, but with the operands reversed.(PowerShell 3.0)
- -notin Like –notcontains, but with the operands reversed.(PowerShell 3.0)

Comparison Operators (example)

- PS C:\> "abc" **-ne** "def"
  True
- PS C:\> 8 **-gt** 6
  True
- PS C:\> "Windows PowerShell" **-like** "*shell"
  True
- PS C:\> "abc", "def" **-Contains** "def"
  True
- PS C:\> "Get-Process" **-Replace** "Get", "Stop“
  Stop-Process
- PS C:\> "book" **-Replace** "B", "C“
  Cook
- PS C:\> 2 **-eq** 3
  False
- PS C:\> "ghi" **-NotIn** "abc", "def“
  True

Variables

- Variables are used to store dynamic or changing data in your PowerShell script.
- A variable can hold a simple number:
  - PS C:\>$mynumber = 42
- Or a string:
  - PS C:\>$mystring = 'The Quick Brown Fox'
- Or an array:
  - PS C:\>$myfolder = dir
- Or any other data

- Once a variable is set it can then be used anywhere a data element is used in PowerShell
- In comparisons
  - PS C:\> $mynumber –eq 12
- In arithmetic
  - PS C:\> $mynumber + 10
- In assignments
  - PS C:\> $mynewstring = $mystring + ' jumped over the lazy dog'
- In Cmdlet/Function calls
  - PS C:\> $myfolder | Write-Output

Flow Control

- Branching
  - If/Then/Else/ElseIf
- Looping
  - While
  - Do While/Do Until
  - For
- Iterating
  - ForEach
  - ForEach-Object
- Skipping, Escaping and Exiting
  - Continue
  - Break
  - Exit

Examples

```
If ( $true –eq $false) { do something } else { do something else }

For ( $counter = 1; $counter –le 10; $counter++ ) { do something}

$counter = 0; do { $counter++; do some work } until { $counter –eq 10 }

Dir | ForEach-Object { do something }

$files = dir
$searchfor = ‘myfilename’
ForEach ($file in $files)
{ if ($file –eq $searchfor) { write-output ‘found it’; break } }
```


LAB 1 – Exploring PowerShell

PowerShell Scripting

Installing the Latest Version of PowerShell (5.x)

- PowerShell is 5.0 is installed by default on Windows 10 and 5.1 is scheduled to be released with Windows Server 2016
- 5.0 is installed as part of Windows Management Framework 5.0
- 5.0 can be installed on
  - Windows Server 2008 R2 SP1, 2012, and 2012 R2
  - Windows 7 SP1, 8.1
- Requires Microsoft .NET Framework 4.5 or above

PowerShell Scripting

- Execution Polices
  - Determine what PowerShell scripts are allowed to run on your computer
  - Options: Restricted (*default*), AllSigned, RemoteSigned, Unrestricted
  - View current policy: PS C:\> Get-ExecutionPolicy
  - Set policy: PS C:\> Set-ExecutionPolicy RemoteSigned
  - **Note:** Execution polices are not meant as a security control. They are supposed to help prevent administrators and users from making unintended mistakes. We can easily bypass them by changing the policy locally or as we call a script.

PowerShell Scripting

- Script Files
  - PowerShell scripts are a flexible and powerful way to perform management and automation tasks
  - Scripts generally have a *.ps1 extension and can be created easily or downloaded from places like the Microsoft Script Center.
  - Bad scripts
    - Be careful as scripts could possibility damage your systems either maliciously or just through coding mistakes.
    - Use trusted sources
    - Run in test first
    - Learn PowerShell syntax so you can “read” a script before running it.

Parameters

- Parameters are used to pass data into functions and scripts.

- For example, if you had a function named SetUserPassword, you might want to pass in the user name and the new password as parameters to tell the function what to operate on.

```
Function SetUserPassword {
    Param ( [string]$username, [string]$password )
    // do something to set the password
}
```

- Parameters can be made mandatory, positional or attached to the pipeline using the optional [Parameter()] attribute:
  - Param([Parameter(Mandatory=$true, Position=1, ValueFromPipeLine=$true)][string]$username)

Output

- Often you will need to output data, status and error messages in some format consumable by the user, either visible on the display or in an output file.
- PS C:\>Write-Output $somevariable - outputs the value of the variable on the display or output stream
- PS C:\>Write-Host $somevariable – outputs the value of the value on the display only
- PS C:\>$somevariable | Out-File $filename – outputs the data piped in to the specified file. Out-File has number of useful parameters you can use such as Append, NoClobber and Force, be sure to check out the online help for these.
- PS C:\>dir | Export-CSV $filename – outputs the data piped in to a CSV formatted file.

Modules

- Modules are used to extend the functionality of PowerShell and group related functions and code together into a easily re-usable package. You can create your own modules, or use the many pre-existing modules that add extended functionality to PowerShell to do things like automated backups, interact with Active Directory or control your VMWare infrastructure.

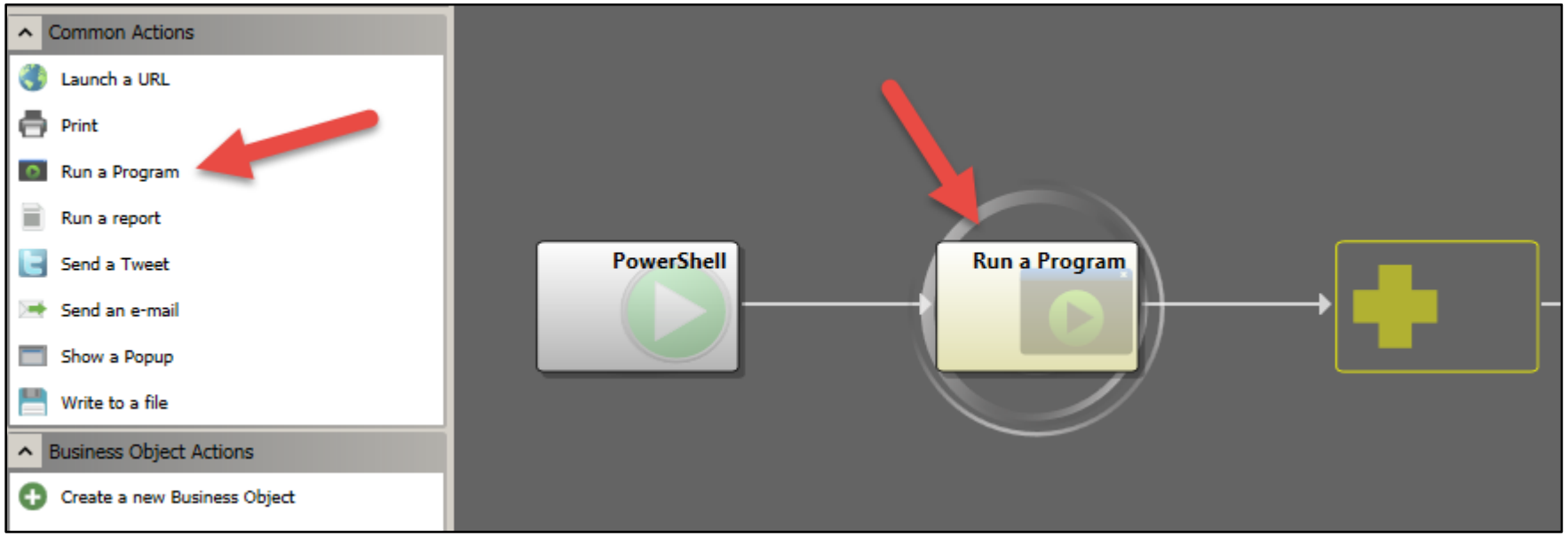
- PS C:\>Get-Module
  - Returns a list of currently loaded modules

- PS C:\>Get-Module –ListAvailable
  - Returns a listing of all modules found on the system

- PS C:\>Import-Module *ModuleName*
  - Loads a module previously installed on the system

LAB 2 – PowerShell Scripts

Calling a Script



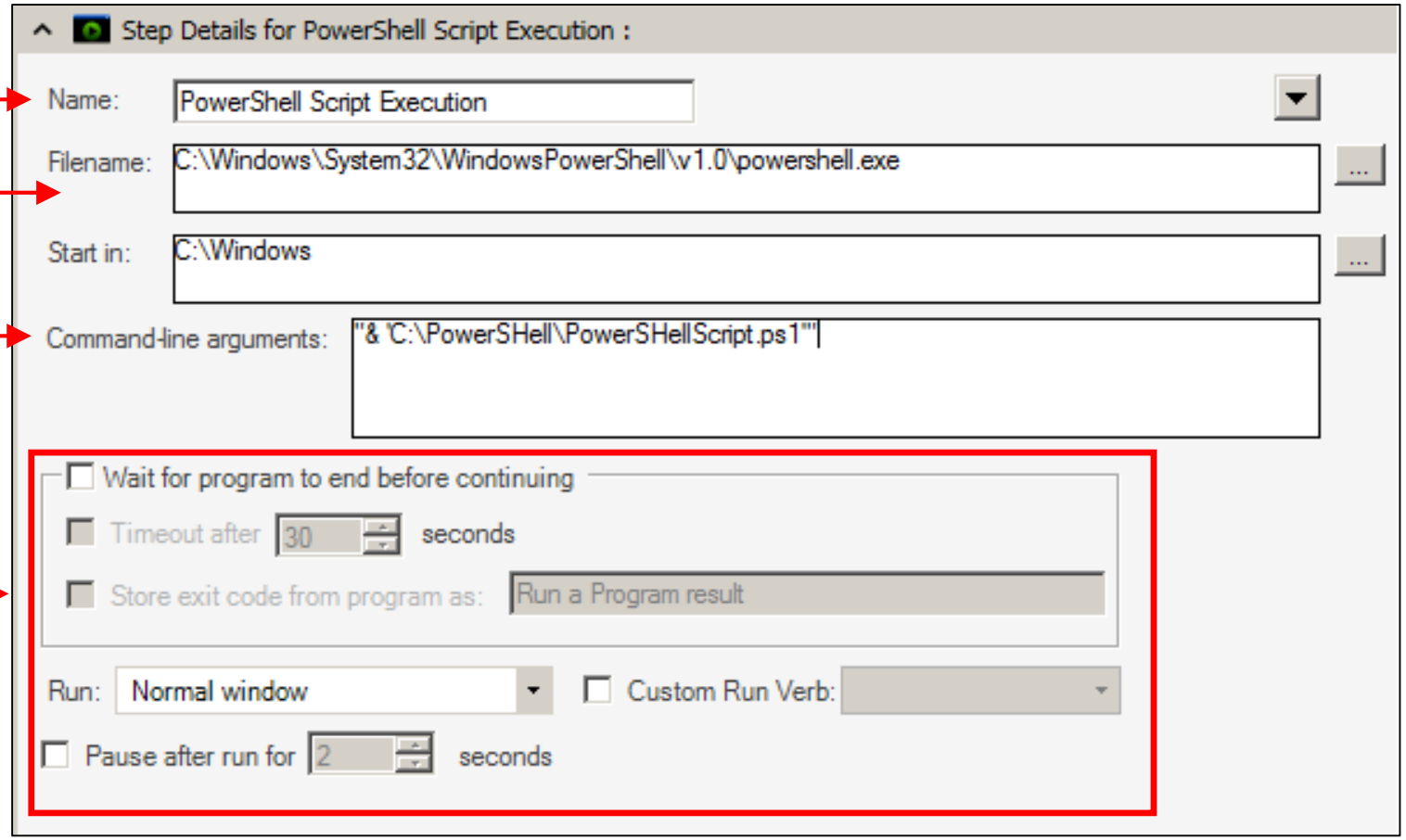
Calling a Script

Display Name of Action

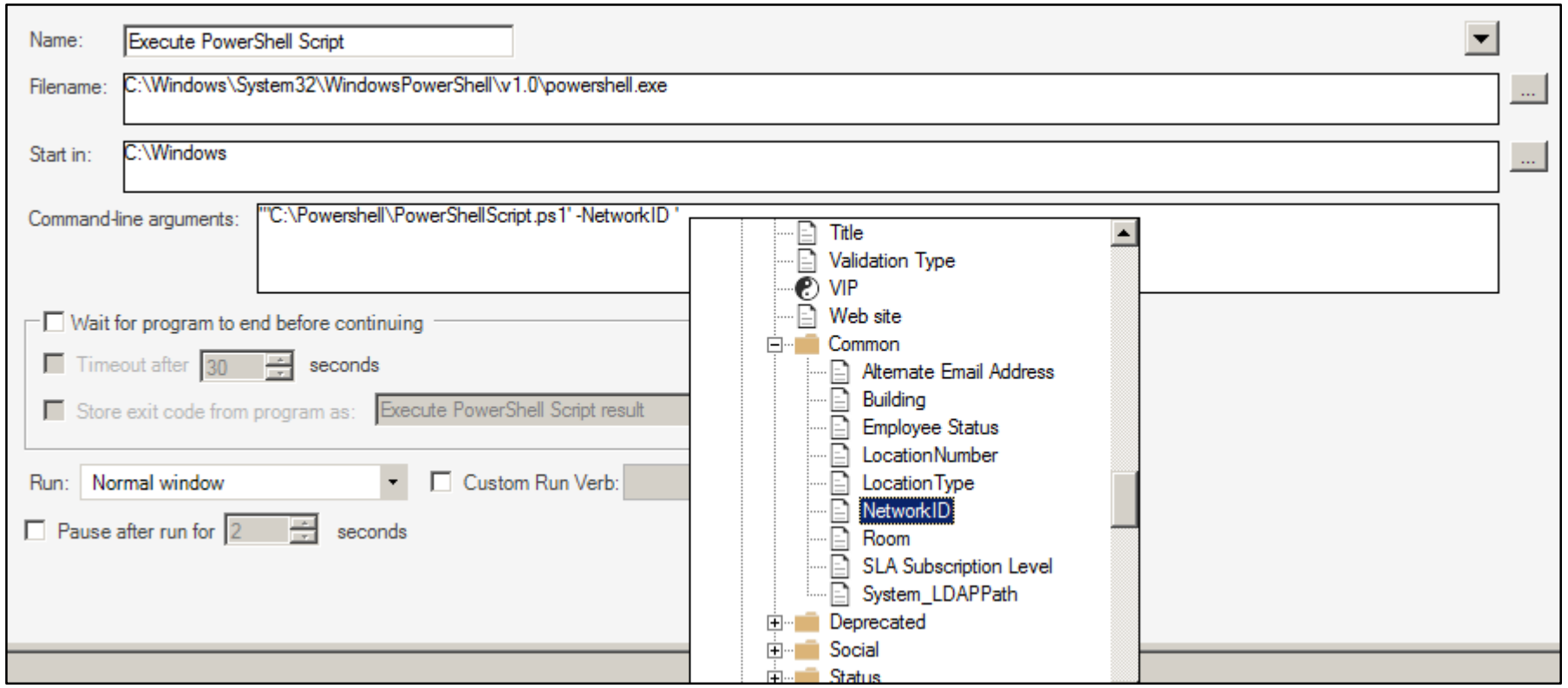
Location of PowerShell Executable

Execution of Script with parameters

Define how One-Step Interacts with Script Execution

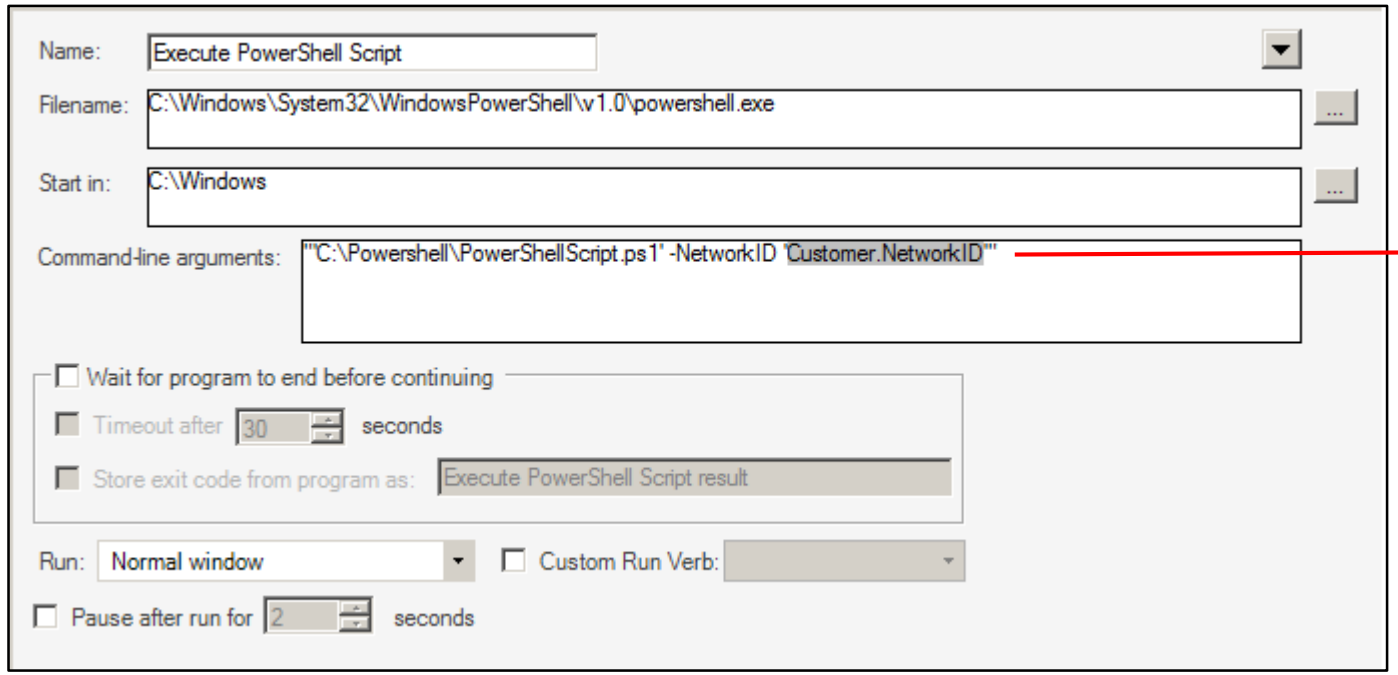


Calling a Script – Setting Parameters



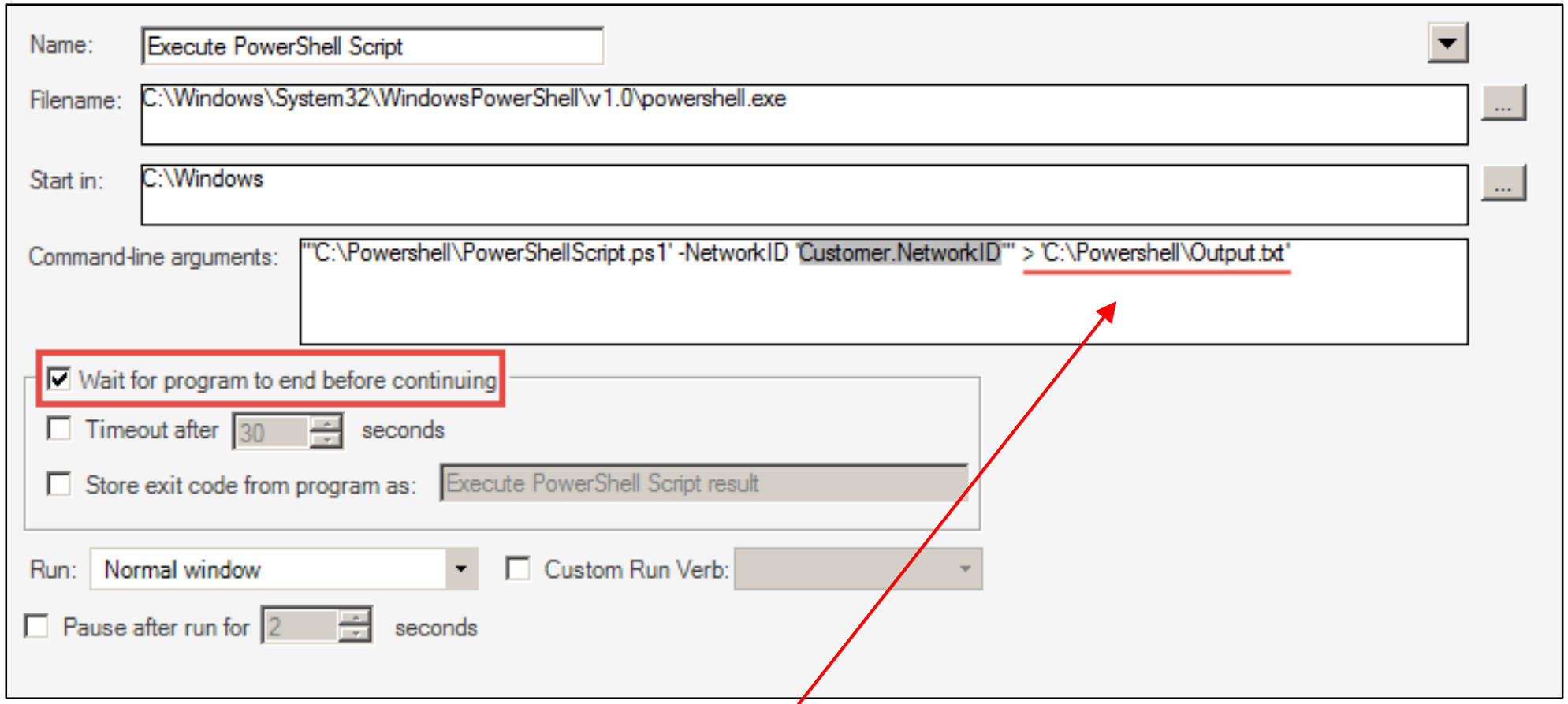
"& 'Script L*ocation' –Parameter1 Value*"

Calling a Script – Setting Parameters



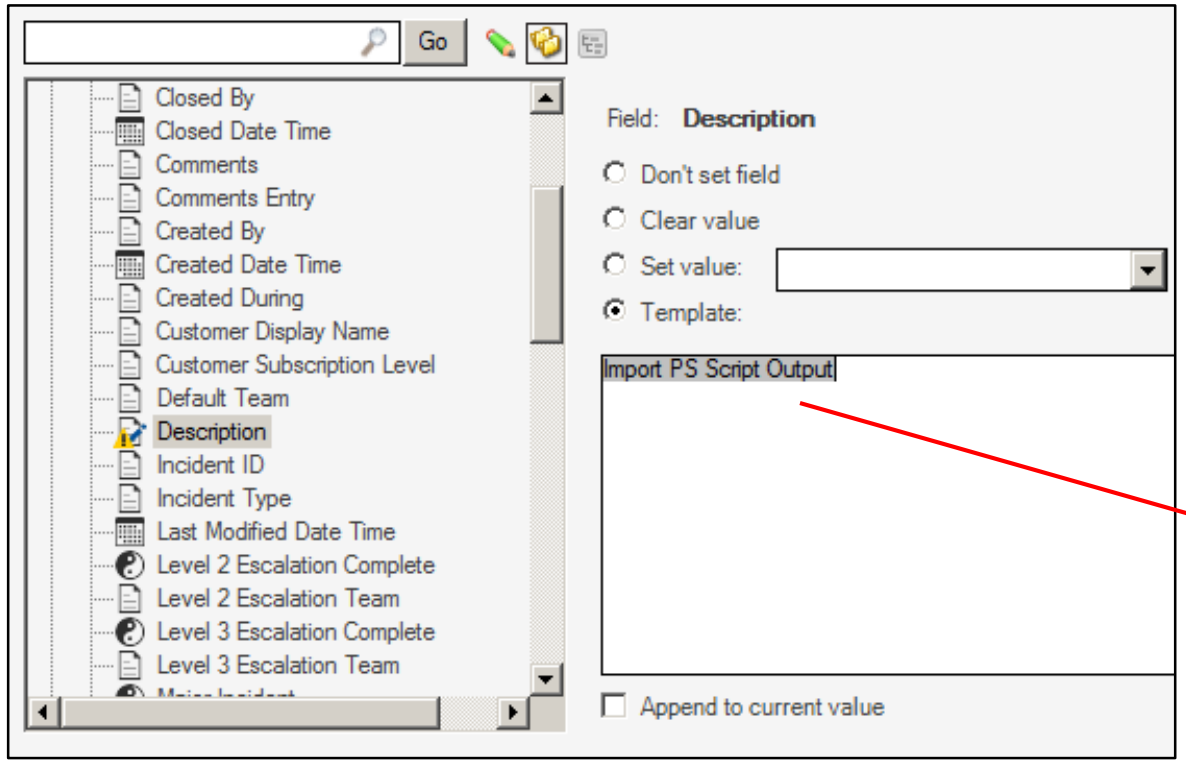

```
1  #Command Line argument handling
2  Param(
3      [string]$networkid
4  )
5
6  Get-ADUser $networkid
7
```

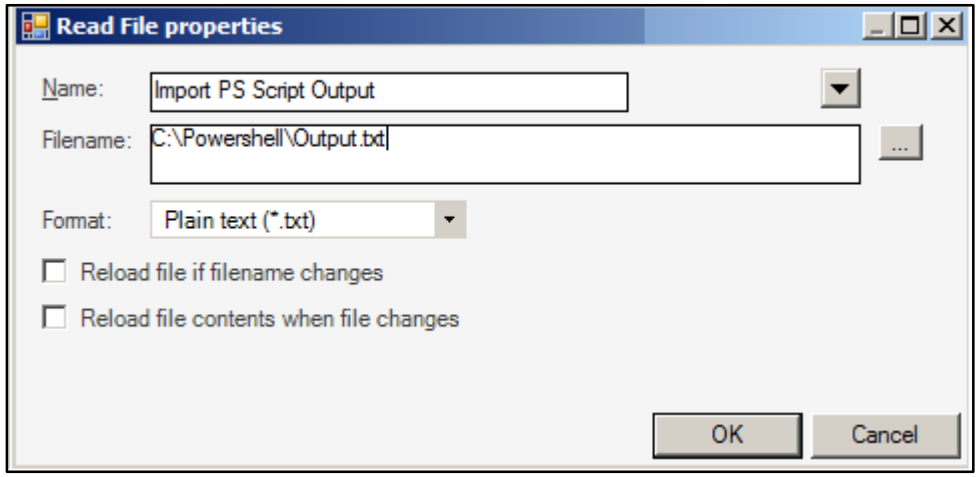
Calling a Script – Output

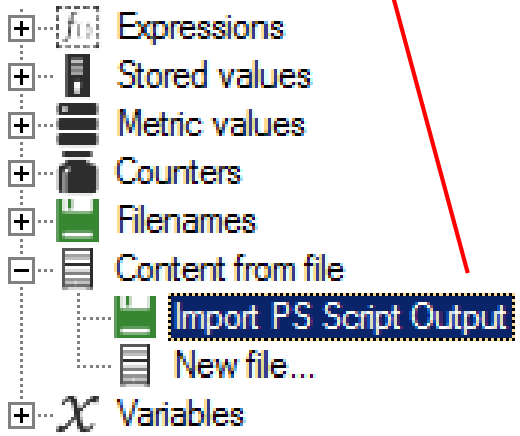


> "<location>\Filename.txt"

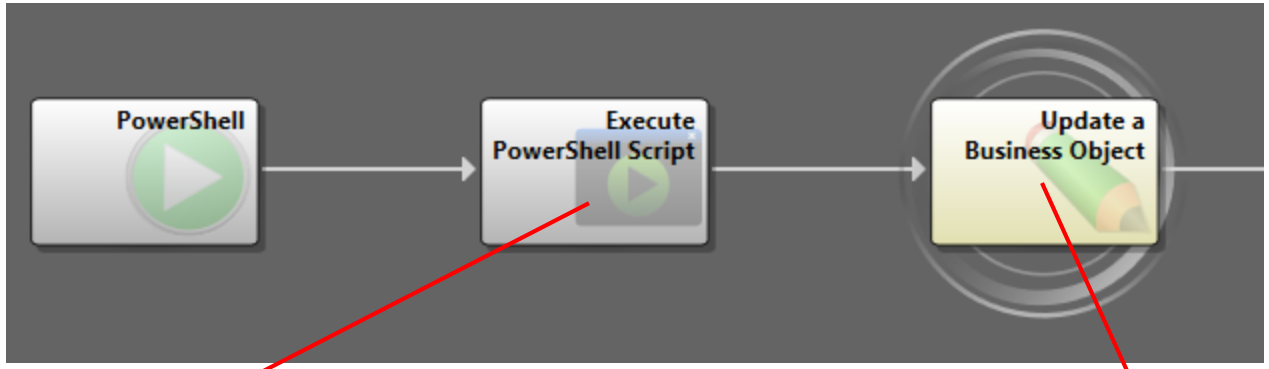
Calling a Script – Output







Calling a Script – Output



Name: Execute PowerShell Script

Filename: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

Start in: C:\Windows

Command-line arguments: "'C:\Powershell\PowerShellScript.ps1' -NetworkID 'Customer.NetworkID'" > 'C:\Powershell\Output.txt'

Wait for program to end before continuing

Timeout after 30 seconds

Store exit code from program as: Execute PowerShell Script result

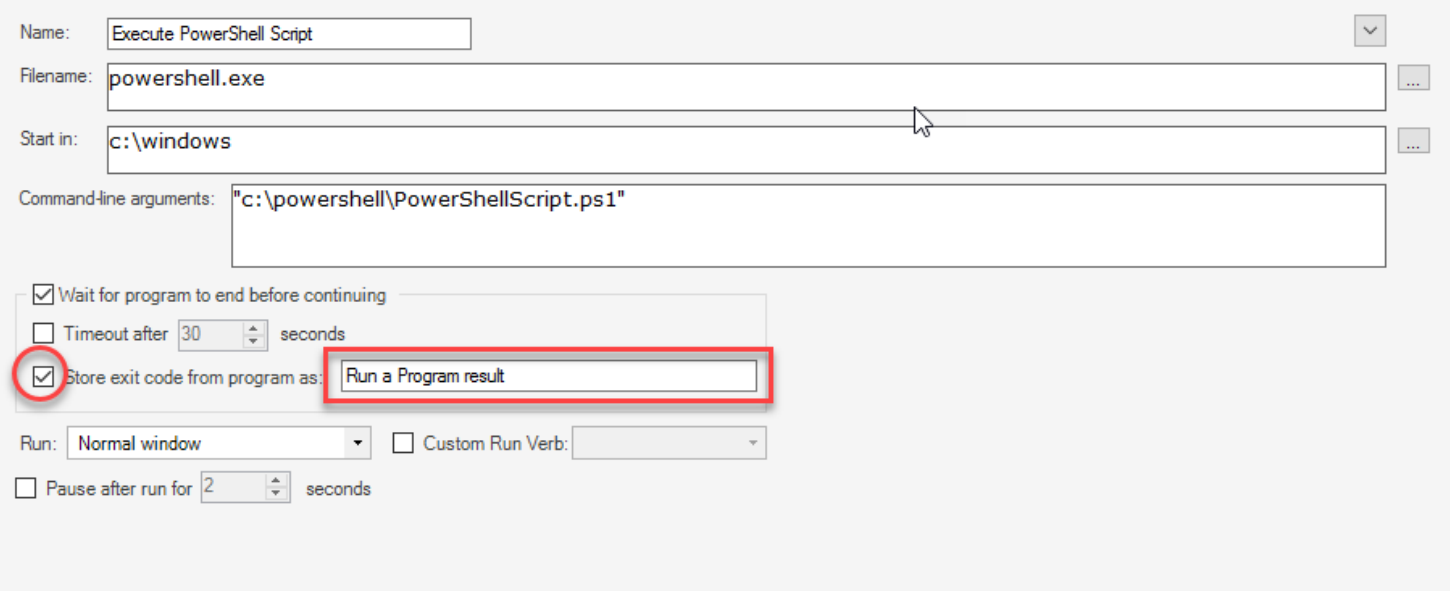
Run: Normal window Custom Run Verb:

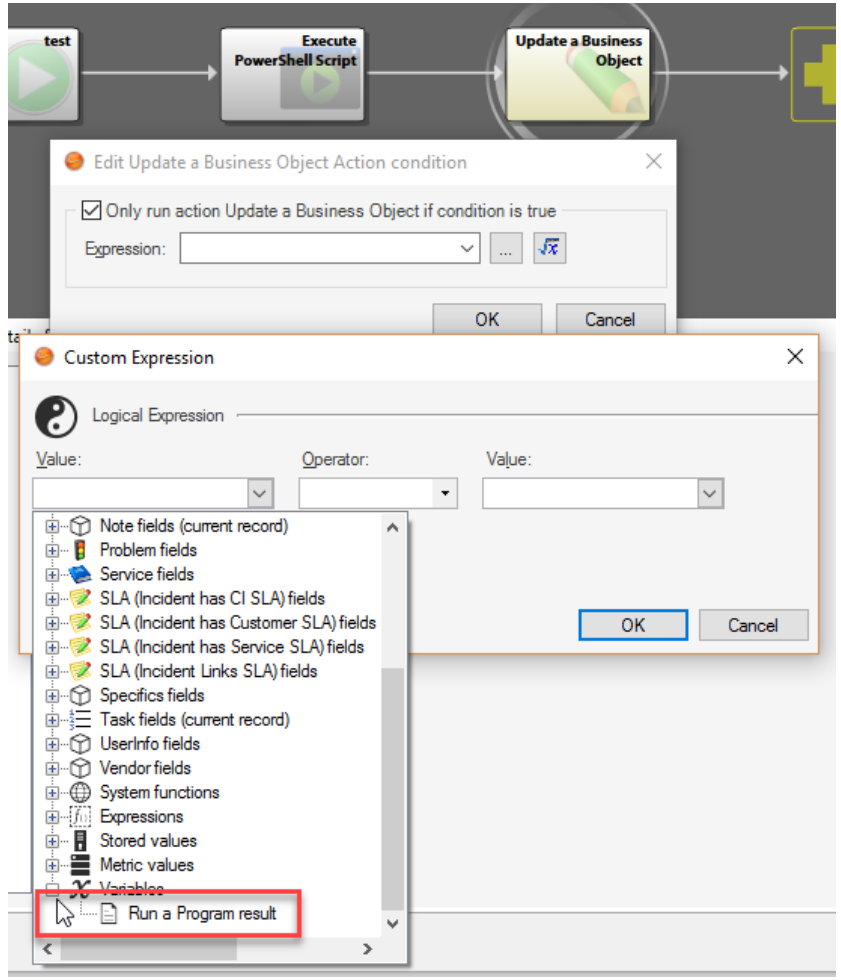
Pause after run for 2 seconds

Please provide any additional details

```
DistinguishedName : CN=Goguen\, Robert,OU=Canada,OU=Users_IOL,DC=iol,DC=irvingoil,DC=net
Enabled           : True
GivenName         : Robert
Name              : Goguen, Robert
ObjectClass       : user
ObjectGUID        : 95f35183-7c28-4e88-821c-b878ea2f2bdf
SamAccountName    : robgog
SID               : S-1-5-21-1979410263-1996570386-1721704976-53998
Surname           : Goguen
UserPrincipalName : robgog@irvingoil.com
```

Handling Errors



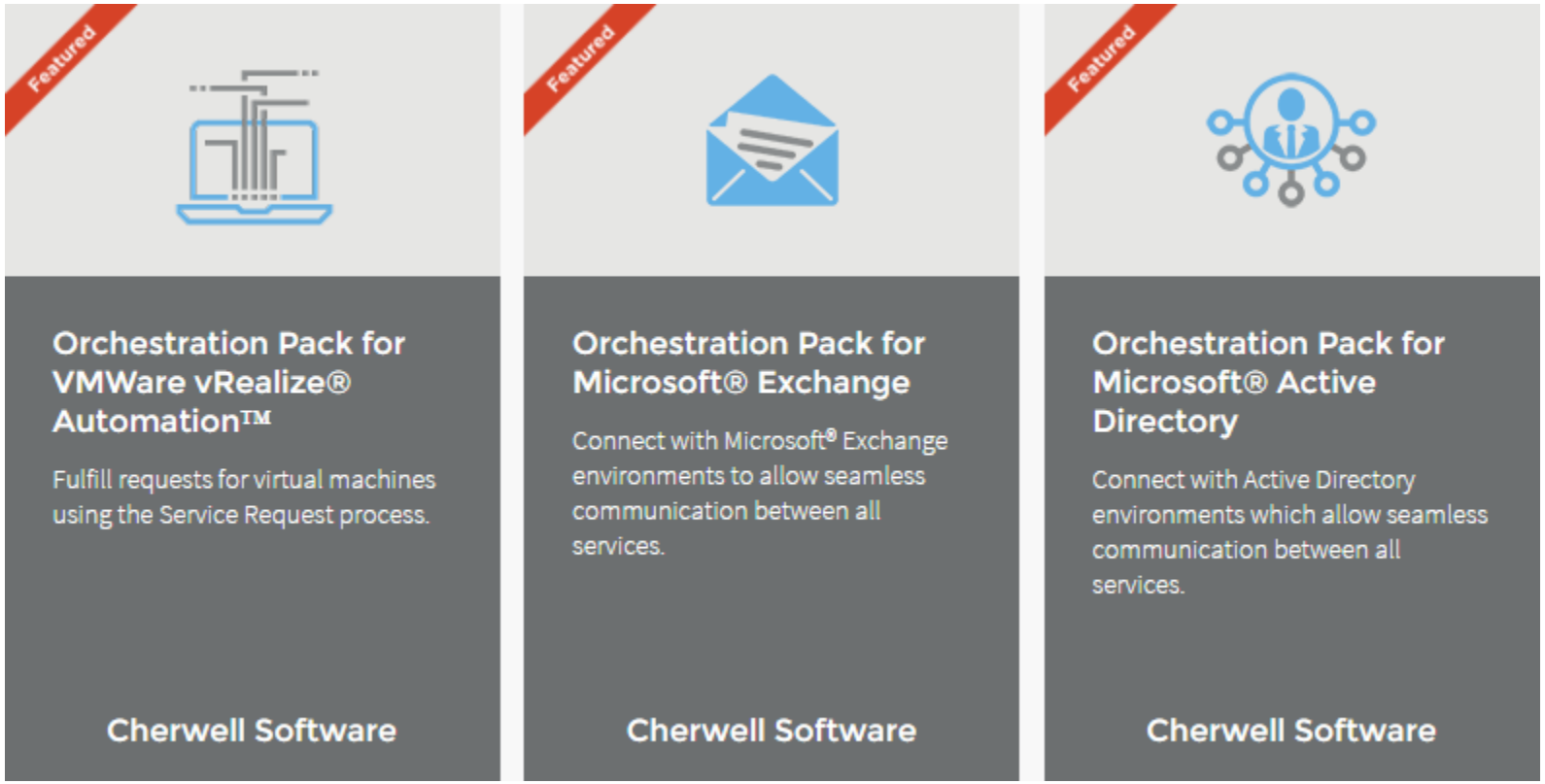


Using Scripting in Cherwell

- Stored Script or On The Fly?
- From Cherwell we have two methods for accessing a script.
- The first method is by calling an existing script stored on the computer's hard drive or on a network share. This requires that the user have access to the file when the script is called otherwise the execution will fail. For complicated scripts, scripts that may change frequently or scripts that need to be secured from prying eyes this may be valid method, but for the most part you'll likely want the other method.
- The other method is to build the script on the fly and store it to a temporary file in your One-Steps using the Write To A File action, then call that file you created from a Run Program action, after the script completes it can be automatically deleted via the One-Step's Write To A File setup.

LAB 3 – Calling PowerShell Scripts from Cherwell One-Steps

Cherwell Orchestration Packs



Cherwell GLOBAL CONFERENCE 2016

Innovation with Purpose.

Thank you for attending this session.
Please fill out an evaluation form.

www.cherwell.com/conference

#CGC16