

SCCM Application Packaging

Part 2 – PowerShell Tips & Tricks

A ZEN GUIDE TO SCCM APPLICATION PACKAGING WITH POWERSHELL

JERRY "ZEN" SENFF

This document is for informational purposes only.

NO WARRANTIES, EXPRESS OR IMPLIED, ARE MADE AS TO THE INFORMATION IN THIS DOCUMENT.

Copyright © 2015 Jerry Senff. All rights reserved.

Document updates copyright © 2019 Jerry Senff. All rights reserved.

Microsoft, Windows, MSDN, and System Center Configuration Manager are trademarks or registered trademarks of Microsoft Corporation in the USA and other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Adjusting Token Privileges in PowerShell © 2010 Precision Computing. Retrieved from <http://www.leeholmes.com/blog/2010/09/24/adjusting-token-privileges-in-powershell/>

Contents

Documentation	6
PowerShell Programing	7
PowerShell Issues	7
PowerShell Execution Policy.....	8
Automating a Manual Package for Task Sequences.....	8
Genericizing Programs with Path Environment Variables	9
Detection Methods	12
File/Folder Permissions	13
Installer	13
Uninstaller.....	13
Local Group Membership	14
Installer	14
Uninstaller.....	14
Test-Path with File Name	15
Installer	15
Uninstaller.....	15
Test-Path with File Name and File Version.....	16
Installer	16
Uninstaller.....	16
Test-Path with Registry Key.....	17
Installer	17
Uninstaller.....	17
Test-Path with Registry Key and Display Version	18
Installer	18
Uninstaller.....	18
Install Methods.....	19
EXE Install.....	20
EXE Install with Parameters	21
MSI Install	22
MSI Install with Transform	23
Uninstall Methods	24
Determining the Uninstall Method to Use	25

Uninstall Methods	27
MsiExec.exe	27
Cmd.exe	29
Case 1	29
Case 2	30
Case 3	31
Case 4	32
Application Uninstall Executable with No Parameters (Start-Process Method)	33
Application Uninstall Executable with Parameters	34
Case 1	34
Case 2	35
Case 3	36
EXE - No Parameters and No User Prompt	37
EXE - No Parameters with User Prompt	38
EXE - Parameters with No User Prompt	40
EXE - Parameters with User Prompt	41
EXE - QuietUninstallString Registry Key	43
EXE - UninstallString Registry Key	44
EXE - UninstallString Registry Key and Double Quotes	45
MSI - No User Prompt	46
MSI - User Prompt	47
MSI - UninstallString Registry Key, Double Quotes, and Switches	49
Other Methods	50
Adding a Path to the PATH Environmental Variable	50
Adjusting Token Privileges	51
SeTakeOwnershipPrivilege	51
SeDebugPrivilege	53
Compatibility Mode	55
Installer	55
Uninstaller	57
Determining an Application's Install Location	60
Enable "Show hidden files, etc" in Folder Options	61
File/Folder ACL Manipulation	62
Installer	62

Uninstaller.....	64
File/Folder Copy Methods	68
Copy-Item Example	68
xcopy Example.....	69
File/Folder Deletion Methods	70
Delete an Install Folder	70
Delete a Start Menu Folder	71
Delete a Desktop Shortcut	72
Granting File/Folder Ownership	73
Group Membership	74
Installer	74
Uninstaller.....	75
Registry Key Deletion.....	77
Shortcuts - Copying and Creating.....	78
Copy an Existing Shortcut Using Copy-Item	78
Creating a Shortcut Using WshShell	79
Stopping Processes and Services	80
Uninstaller.....	82
32-Bit ODBC Connections	84
Installer	85
Uninstaller.....	86

Documentation

The documentation has been broken into three components to make both reading and downloading from GitHub easier:

- Part 1 covers the application packaging process and the SCCM interface.
- Part 2 covers the PowerShell programming tips and tricks used with SCCM.
- Part 3 is a collection of example PowerShell install and uninstall programs demonstrating the PowerShell tips and tricks used with SCCM for the successful installation of complex software packages.

PowerShell Programing

PowerShell Issues

Some issues regarding programing with PowerShell and SCCM:

- **Creating the Check:** Before any action is taken by a PowerShell program, a check must first be made to ensure that the action needs to be performed or skipped. For example, if the program goes to create something that already exists, PowerShell throws an error. To not have any errors thrown, the appropriate check must be created to ensure that the action is performed if needed or skipped if already in place. The basic format for the install and uninstall PowerShell checks are as follows:

Application install check example

```
If uninstall registry key does not exist
{
    Install the application
}
Else
{
    Application already installed - skip step
}
```

Application uninstall check example

```
If uninstall registry keys exists
{
    Uninstall the application
}
Else
{
    Application not installed - skip step
}
```

- **Folder/File Permissions:** The PowerShell `Set-Acl` function has an "undocumented" feature where the function attempts to assign ownership of the file or folder having the ACL change to the user account calling the function if no previous ownership exists. In the case of SCCM, where everything runs in the SYSTEM context, the `Set-Acl` function will fail due to the built-in Windows limitation where the SYSTEM account cannot "own" any files or folders. The way to get around this problem is to first assign the Administrators group ownership of the folder using the `takeown /F folderPath /R /A` PowerShell command before trying to change the ACLs using the `Set-Acl` function..
- **Windows System Folder/File Permissions:** System folders and files are even more problematic to deal with. To effectively manipulate the ACLs, one must first call a special piece of code that enables the `SeTakeOwnershipPrivilege` token privilege which basically gives the calling PowerShell program full ownership of the system's files, folders, and registry for the time that the PowerShell program runs.

Note: The program used to enable the `SeTakeOwnershipPrivilege` is from Precision Computing (copyright 2015). Please include the commented reference (below) crediting them for their excellent work in creating this fine piece of code. Comments are also included in the code snippets in the **Adjusting Token Privileges** section and examples.

```
# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs without errors      #
# Privilege token only good for current PowerShell session                        #
#                                                                                #
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2010 #
# http://www.leeholmes.com/blog/2010/09/24/adjusting-token-privileges-in-powershell/ #
```

PowerShell Execution Policy

In SCCM, use the following method to call PowerShell programs to get the user elevation needed to call some methods:

```
Powershell.exe -executionpolicy Bypass -file "FileName"
```

Examples:

```
Powershell.exe -executionpolicy Bypass -file "EasyLobby10-Install.ps1"
```

```
Powershell.exe -executionpolicy Bypass -file "EasyLobby10-uninstall.ps1"
```

Automating a Manual Package for Task Sequences

If an application absolutely needs to be automated to be included in a task sequence, it is possible to do so by programming the installer. Most times, a package is set to manually install because no shortcuts and/or a Start Menu group does not get created during a silent install due to the installer not interacting with the desktop.

To get by the silent install method limitation, do the following:

- Install the application on a target system.
- Copy any desktop shortcuts and the **Start Menu** group to a folder in the application's Software Vault folder named **Shortcuts**.
- Create a PowerShell program to install the application silently.
- In the PowerShell install program:
 - Add code to copy the desktop shortcuts from the SCCM cache folder to the **All Users** desktop.
 - Add code to copy the **Start Menu** folder from the SCCM cache folder to the **All Users Start Menu**.
- Open the properties for the application's SCCM package deployment type
 - On the **Programs** tab, change the **Installer program** to the PowerShell installer program with the **-ExecutionPolicy Bypass** switch:

```
Powershell.exe -executionpolicy bypass -file "SomeApp-Install.ps1"
```


Genericizing Programs with Path Environment Variables

Windows system environment path variables are used in PowerShell install and uninstall programs to lessen the number of hard-coded paths in the programs. The path environment variables are always in the form of `${env:ProgramFiles}` as in the example below:

```
$AppName1Uninstall = ${env:ProgramFiles} + '\AppName1\Uninstaller.EXE'
```

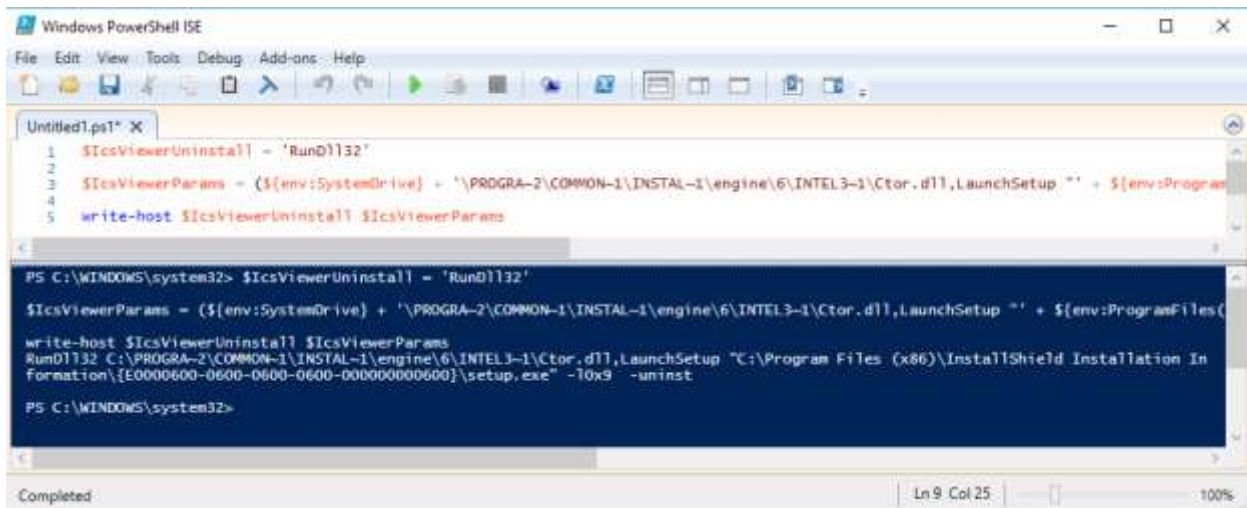
For paths with spaces in them, double quotes (highlighted in yellow) will be required at the beginning and the end of the string, as below:

```
$AppName1Uninstall = '"' + ${env:ProgramFiles} + '\AppName1\Uninstaller.EXE'"'
```

Parameter string variables that contain paths in their strings, like RunDLL32 uninstall parameter strings, will also need to have those paths genericized as in the following *ICS Viewer* example:

```
$IcsViewerUninstall = 'RunDll32'
$IcsViewerParams = (${env:SystemDrive} +
'\PROGRA~2\COMMON~1\INSTAL~1\engine\6\INTEL3~1\Ctor.dll,LaunchSetup '" +
${env:ProgramFiles(x86)} + '\InstallShield Installation
Information\{E0000600-0600-0600-0600-000000000600}\setup.exe"' -10x9 -
uninst')
```

Always watch out for placement of the double quotes around paths containing spaces. To check the output before running a program, copy the two string variables and paste them into a PowerShell console and press the **Enter** key. In the console window, execute a **write-host** command with the two string variables separated by a space.



```
Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Untitled1.ps1 X
1 $IcsViewerUninstall = 'RunDll32'
2
3 $IcsViewerParams = (${env:SystemDrive} + '\PROGRA~2\COMMON~1\INSTAL~1\engine\6\INTEL3~1\Ctor.dll,LaunchSetup '" +
4 ${env:ProgramFiles(x86)} + '\InstallShield Installation Information\{E0000600-0600-0600-0600-000000000600}\setup.exe"' -10x9 -
5 uninst')

PS C:\WINDOWS\system32> $IcsViewerUninstall = 'RunDll32'
$IcsViewerParams = (${env:SystemDrive} + '\PROGRA~2\COMMON~1\INSTAL~1\engine\6\INTEL3~1\Ctor.dll,LaunchSetup '" +
write-host $IcsViewerUninstall $IcsViewerParams
RunDll32 C:\PROGRA~2\COMMON~1\INSTAL~1\engine\6\INTEL3~1\Ctor.dll,LaunchSetup "C:\Program Files (x86)\InstallShield Installation In
formation\{E0000600-0600-0600-0600-000000000600}\setup.exe" -10x9 -uninst
PS C:\WINDOWS\system32>

Completed Ln 9 Col 25 100%
```

Open the corresponding application package document page and copy the PowerShell console window result and the actual **UninstallString** from the Notes section in to the PowerShell program pane and compare the two strings. Correct any errors and check the strings again.

To find out what path environment variables are available, open a *PowerShell ISE* window with Administrator privileges (**Run ISE as Administrator**), switch to the ENV: drive, and list the directory.

The most commonly used path environment variables are as follows:

Name	Value
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\<user_name>\AppData\Roaming
LOCALAPPDATA	C:\Users\<user_name>\AppData\Local
ProgramData	C:\ProgramData
ProgramFiles	C:\Program Files
ProgramFiles(x86)	C:\Program Files (x86)
PUBLIC	C:\Users\Public
SystemDrive	C:
SystemRoot	C:\WINDOWS
TEMP	C:\Users\<user_name>\AppData\Local\Temp
USERPROFILE	C:\Users\<user_name>
windir	C:\WINDOWS

Detection Methods

The following snippets are the check methods used to verify that an install or uninstall step needs to be taken. See the relevant subsections in the **Install Methods**, **Uninstall Methods**, and **Other Methods** sections for PowerShell code examples for accomplishing different tasks.

File/Folder Permissions

Add the *Adjusting Token Privileges* code at the beginning of a PowerShell program to enable the **SeTakeOwnershipPrivilege** for modifying ACLs on system files and folders.

Installer

```
# Installer: Check for user permissions on file or folder
# Variables -- Modify
$folder1 = (${env:ProgramFiles(x86)} + '\AppName1')

# MODIFY - Administrators, Users, Power Users, DOMAIN\SecurityGroup, etc.
$User = 'BUILTIN\Users'

# MODIFY - FullControl permission does not need 'Synchronize' for string matching
$FolderRightsLong = 'write, Synchronize'

# MODIFY - Allow, Deny
$AllowDeny = 'Allow'

# Check if $User has $AllowDeny $FolderRightsLong to $folder1
$r = (Get-Acl $folder1).Access | where {$_.IdentityReference -eq $User -and
    $_.FileSystemRights -eq $FolderRightsLong -and $_.AccessControlType -eq
    $AllowDeny}

If ($r -eq $null)
{
    # Grant $User $AllowDeny $FolderRightsLong to $folder1
}
Else
{
    # $User has $AllowDeny $FolderRightsLong to $folder1
}
```

Uninstaller

```
# Uninstaller: Check for user permissions on file or folder
# Variable -- Modify
$folder1 = (${env:ProgramFiles(x86)} + '\AppName1')

# MODIFY - Administrators, Users, Power Users, DOMAIN\SecurityGroup, etc.
$User = 'BUILTIN\Users'

# MODIFY - FullControl permission does not need 'Synchronize' for string matching
$FolderRightsLong = 'write, Synchronize'

# MODIFY - Allow, Deny
$AllowDeny = 'Allow'

# Check if $User has $AllowDeny $FolderRightsLong to $folder1
$r = (Get-Acl $folder1).Access | where {$_.IdentityReference -eq $User -and
    $_.FileSystemRights -eq $FolderRightsLong -and $_.AccessControlType -eq
    $AllowDeny}

If ($r -ne $null)
{
    # Remove $User $AllowDeny $FolderRightsLong to $folder1
}
Else
{
    # $User does not have $AllowDeny $FolderRightsLong to $folder1
}
```

Local Group Membership

Installer

```
# Installer: Check for domain group in local group

# Variables -- Modify
$DomainGroup = 'DomainGroupname'
$Domain = 'DomainName'
$LocalGroup = 'LocalGroupName'    # Administrators, Users, Power Users, etc

# Check for domain group in local group
$r = (([ADSI]"winNT://./$LocalGroup").Invoke("IsMember",
    "winNT://$Domain/$DomainGroup"))

If ($r -match 'False')
{
    # Add $DomainGroup to $LocalGroup
}
Else
{
    # $DomainGroup is already a member of $LocalGroup
}
```

Uninstaller

```
# Uninstaller: Check for domain group in local group

# Variables -- Modify
$DomainGroup = 'DomainGroupname'
$Domain = 'DomainName'
$LocalGroup = 'LocalGroupName'    # Administrators, Users, Power Users, etc

# Check for domain group in local group
$r = (([ADSI]"winNT://./$LocalGroup").Invoke("IsMember",
    "winNT://$Domain/$DomainGroup"))

If ($r -match 'True')
{
    # Remove $DomainGroup from $LocalGroup
}
Else
{
    # $DomainGroup is not a member of $LocalGroup
}
```

Test-Path with File Name

Installer

```
# Installer: Test-Path Check with File Name Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1FilePath = (${env:ProgramFiles(x86)} + '\AppName1\AppName1.exe')

# Test if AppName1 file name exists
If ((Test-Path -Path $AppName1FilePath) -ne 'True')
{
    # Install $AppName1
}
Else
{
    # $AppName1 already installed
}
```

Uninstaller

```
# Uninstaller: Test-Path Check with File Name Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1FilePath = (${env:ProgramFiles(x86)} + '\AppName1\AppName1.exe')

# Test if AppName1 file name exists
If ((Test-Path -Path $AppName1FilePath) -eq 'True')
{
    # Uninstall $AppName1
}
Else
{
    # $AppName1 not installed
}
```

Test-Path with File Name and File Version

Installer

```
# Installer: Test-Path Check with File Name, File Version Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1FilePath = (${env:ProgramFiles(x86)} + '\AppName1\AppName1.exe')
$AppName1FileVersion = '1.2.3.4'

# Test if AppName1 file name exists
If ((Test-Path -Path $AppName1FilePath) -eq 'True')
{
    # Application installed, now check for proper file version
    If ((Get-Item -Path $AppName1FilePath).VersionInfo.FileVersion -ne
        $AppName1FileVersion)
    {
        # Update $AppName1 to correct version
    }
    Else
    {
        # $AppName1 version is correct
    }
}
Else
{
    # $AppName1 not installed
}
```

Uninstaller

```
# Uninstaller: Test-Path Check with File Name, File Version Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1FilePath = (${env:ProgramFiles(x86)} + '\AppName1\AppName1.exe')
$AppName1FileVersion = '1.2.3.4'

# Test if AppName1 file name with file version exists
If (((Test-Path -Path $AppName1FilePath) -eq 'True') -and ((Get-Item -Path
    $AppName1FilePath).VersionInfo.FileVersion -eq $AppName1FileVersion))
{
    # Uninstall $AppName1
}
Else
{
    # $AppName1 not installed
}
```


Test-Path with Registry Key

Installer

```
# Installer: Test-Path Check with Registry Key Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey =
    'HKLM:SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Test if AppName1 uninstall registry key exists
If ((Test-Path -Path $AppName1Regkey) -ne 'True')
{
    # Install $AppName1
}
Else
{
    # $AppName1 already installed
}
```

Uninstaller

```
# Uninstaller: Test-Path Check with Registry Key Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey =
    'HKLM:SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Test if AppName1 uninstall registry key exists
If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    # Uninstall $AppName1
}
Else
{
    # $AppName1 not installed
}
```

Test-Path with Registry Key and Display Version

Installer

```
# Installer: Test-Path Check with Registry Key, Display Version Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey =
'HKLM:SOFTWARE\wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'
$AppName1DisplayVersion = '1.2.3.4'

# Test if AppName1 uninstall registry key exists
If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    # Application installed, now check for proper version
    If ((Get-ItemProperty -Path $AppName1Regkey).DisplayVersion -eq
        $AppName1DisplayVersion)
    {
        # $AppName1 version is correct
    }
    Else
    {
        # Update $AppName1 to correct version
    }
}
Else
{
    # $AppName1 not installed
}
```

Uninstaller

```
# Uninstaller: Test-Path Check with Registry Key, Display Version Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey =
'HKLM:SOFTWARE\wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'
$AppName1DisplayVersion = '1.2.3.4'

# Test if AppName1 uninstall registry key exists
If (((Test-Path -Path $AppName1Regkey) -eq 'True') -and ((Get-ItemProperty -Path
    $AppName1Regkey).DisplayVersion -eq $AppName1DisplayVersion))
{
    # Uninstall $AppName1
}
Else
{
    # $AppName1 $AppName1DisplayVersion not installed
}
```

Install Methods

The following PowerShell techniques and methods can be used for application installation and configuration (see relevant sections):

- Detection methods
 - Check if the application is installed using:
 - Test-Path with file name
 - Test-Path with file name and file version
 - Test-Path with **Uninstall** registry key
 - Test-Path with **Uninstall** registry key and display version
 - Installed component(s) check using any of the four above checks
 - File/folder ACL permission check
 - Local group membership check
- Install methods
 - .EXE install
 - .EXE install with parameters
 - .MSI install
 - .MSI install with transform
- Other methods
 - Adding a path to the PATH Environmental Variable
 - Adjusting token privileges
 - Enable **SeTakeOwnershipPrivilege** for system file and folder ACL permission changes
 - Enable **SeDebugPrivilege** for installing SQL Server when blocked by Group Policy
 - Compatibility mode, including RUNASADMIN
 - Determine an application's install location
 - Enable **Show hidden files, etc...** in Windows Explorer **Folder Options**
 - File/folder ACL manipulation to grant permissions to local user group(s)
 - File/Folder copy methods using Copy-Item and xcopy
 - Grant folder ownership to Administrator prior to ACL permission changes
 - Group membership - adding a domain security group to local user group
 - Shortcuts - copying with Copy-Item and creating with WshShell
 - 32-bit ODBC connections - creating from registry key screenshots

EXE Install

```
# EXE Installer Example - No Parameters

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Install = '.\SomeApplication.exe'
$AppName1Regkey = 'HKLM:\SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = ($env:SystemDrive + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Installer"
Write-Host ""
Write-Host "Purpose: Install the following components:"
Write-Host "    - Install $AppName1"
Write-Host ""

# Install Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -ne 'True')
{
    Write-Host "    $AppName1 is not installed."
    Write-Host ""
    Write-Host "Installing $AppName1..."
    Write-Host ""
    Write-Host "Command: $AppName1Install"
    Start-Process -FilePath $AppName1Install -ErrorVariable +err -Verb Open -Wait
    Write-Host ""
    Write-Host "    $AppName1 install complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 already installed."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

EXE Install with Parameters

```
# EXE Installer Example with Parameters

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Install = '.\SomeApplication.exe'
$AppName1Params = '/S /v/qn'
$AppName1Regkey = 'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = ($env:SystemDrive + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Installer"
Write-Host ""
Write-Host "Purpose: Install the following components:"
Write-Host "    - Install $AppName1"
Write-Host ""

# Install Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -ne 'True')
{
    Write-Host "    $AppName1 is not installed."
    Write-Host ""
    Write-Host "Installing $AppName1..."
    Write-Host ""
    Write-Host "Command: $AppName1Install $AppName1Params"
    Start-Process -FilePath $AppName1Install -ArgumentList $AppName1Params -
        ErrorVariable +err -Verb Open -Wait
    Write-Host ""
    Write-Host "    $AppName1 install complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 already installed."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

MSI Install

```
# MSI Installer Example

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Msi = '.\SomeApplication.msi'
$AppName1Regkey = 'HKLM:\SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Variables -- Do NOT modify
$MsiExec = ($env:SystemRoot + '\System32\msiexec.exe')
$AppName1Params = '/i "' + $AppName1Msi + '" /qb/ norestart' # or /qn

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = ($env:SystemDrive + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Installer"
Write-Host ""
Write-Host "Purpose: Install the following components:"
Write-Host "    - Install $AppName1"
Write-Host ""

# Install Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -ne 'True')
{
    Write-Host "    $AppName1 is not installed."
    Write-Host ""
    Write-Host "Installing $AppName1..."
    Write-Host ""
    Write-Host "Command: $MsiExec $AppName1Params"
    Write-Host ""
    Start-Process -FilePath $MsiExec -ArgumentList $AppName1Params -ErrorVariable +err
    -Verb Open -wait
    Write-Host ""
    Write-Host "    $AppName1 install complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 is already installed."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

MSI Install with Transform

```
# MSI Installer Example with Transform

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Msi = '.\SomeApplication.msi'
$AppName1Mst = '.\SomeApplication.mst'
$AppName1Regkey = 'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Variables -- Do NOT modify
$MsiExec = ($env:SystemRoot + '\System32\msiexec.exe')
$AppName1Params = '/i "' + $AppName1Msi + '" /t "' + $AppName1Mst + '" /qb/
norestart' # or /qn

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\ ' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = ($env:SystemDrive + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Installer"
Write-Host ""
Write-Host "Purpose: Install the following components:"
Write-Host "        - Install $AppName1"
Write-Host ""

# Install Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -ne 'True')
{
    Write-Host "    $AppName1 is not installed."
    Write-Host ""
    Write-Host "Installing $AppName1..."
    Write-Host ""
    Write-Host "Command: $MsiExec $AppName1Params"
    Write-Host ""
    Start-Process -FilePath $MsiExec -ArgumentList $AppName1Params -ErrorVariable +err
    -Verb Open -Wait
    Write-Host ""
    Write-Host "    $AppName1 install complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 is already installed."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

Uninstall Methods

The following PowerShell techniques and functions can be used for application and configuration removal (see relevant subsections):

- Detection methods
 - Check for installed application(s) using:
 - Test-Path with file name
 - Test-Path with file name and file version
 - Test-Path with **Uninstall** registry key
 - Test-Path with **Uninstall** registry key and display version
 - Installed component(s) check using any of the four above checks
 - File/folder ACL permission check
 - Local group membership check
- Uninstall methods - Read the **Determining the Uninstall Method to Use** section below first!
 - .EXE with no parameters and no user prompt
 - .EXE with no parameters and with user prompt
 - .EXE with parameters and no user prompt
 - .EXE with parameters and with user prompt
 - .EXE with **QuietUninstallString** registry key
 - .EXE with **UninstallString** registry key
 - .EXE with **UninstallString** registry key with quotes
 - .MSI with no user prompt
 - .MSI with user prompt
 - .MSI with **UninstallString** registry key, quotes, and switches
- Other methods
 - Adjusting token privileges
 - Enable **SeTakeOwnershipPrivilege** for system file and folder ACL permission changes
 - Compatibility mode setting removal
 - File/folder ACL manipulation
 - File/folder deletion methods
 - Delete an install folder
 - Delete a **Start Menu** folder
 - Delete a shortcut from the default user's **Desktop**
 - Group membership - Removing a domain security group from a local user group with user prompt
 - Deleting leftover **Uninstall** registry keys
 - Stopping processes and services before uninstalling an application
 - 32-bit ODBC connection deletion with user prompt

Determining the Uninstall Method to Use

The uninstall method used depends on the format of the **UninstallString**, or **QuietUninstallString**, in an application's uninstall registry key.

All uninstall methods use the PowerShell **Start-Process** command with the uninstall executable and any parameters fed into the command using the following format:

```
$AppName1Uninstall = ${env:ProgramFiles(x86)} + 'SomeApplication\Uninstall.exe'
$AppName1Params = '/SILENT'

Start-Process -FilePath $AppName1Uninstall -ArgumentList $AppName1Params -
    ErrorVariable +err -Verb Open -Wait
```

Start-Process switches:

- **FilePath:** The full path to the application's uninstall executable using system environment path variables. Executables can be `MsiExec.exe`, `Cmd.exe`, `AppName1Uninstall.exe`, `RunDLL32.exe`, etc.
- **ArgumentList:** The uninstall parameters passed into the command via a string variable.
- **ErrorVariable:** The `+err` flag adds errors to an error log defined at the top of the uninstall program:

```
# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)
```

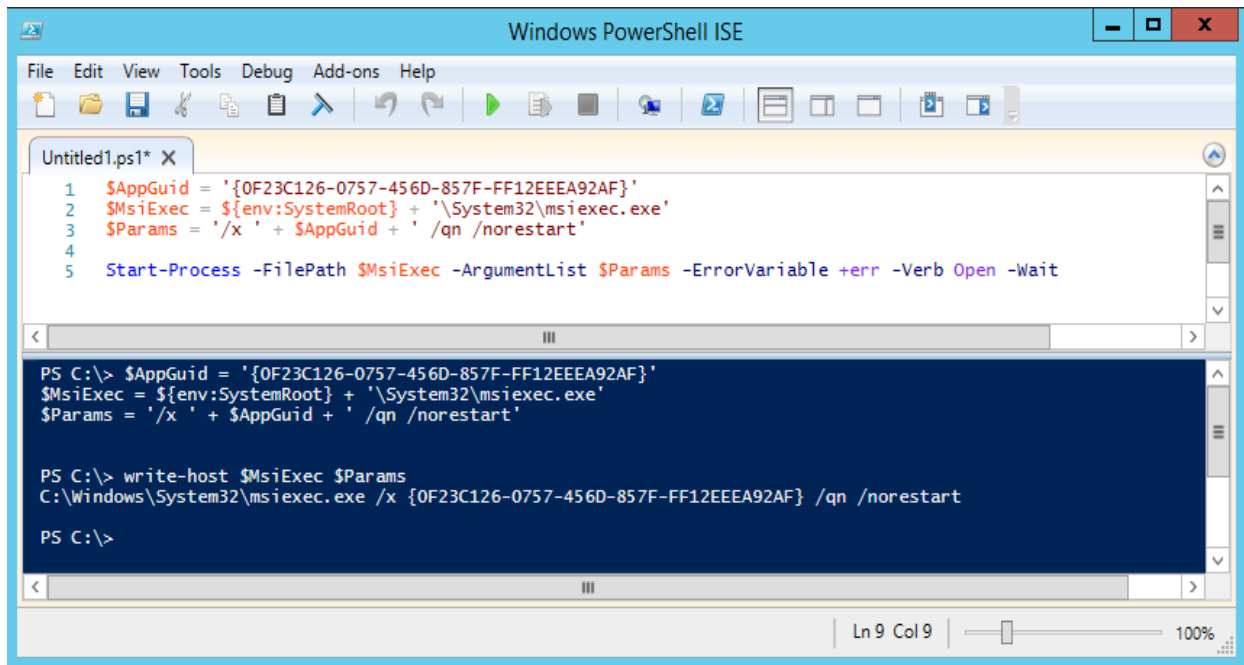
- **Verb:** Specifies a verb to use when starting the process. The verbs that are available are determined by the file name extension of the file that runs in the process. SCCM PowerShell programs normally use the `-Verb Open` switch.

File type Verbs

```
-----
.cmd-----Edit, Open, Print, RunAs
.exe-----Open, RunAs
.txt-----Open, Print, PrintTo
.wav-----Open, Play
```

- **Wait:** Waits for the specified process to complete before accepting more input.

To check the output before running a program, copy the two string variables and past them into *Windows PowerShell ISE* console pane and press the **Enter** key. In the console pane, run a `write-host` command with the two string variables separated by a space as demonstrated by the example below.



The screenshot shows the Windows PowerShell ISE interface. The top menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. Below the menu is a toolbar with various icons. The main editor pane shows a script in a file named 'Untitled1.ps1'. The script contains five lines of PowerShell code. The console pane below the editor shows the execution of the script, with the output of the `write-host` command displayed.

```
1 $AppGuid = '{0F23C126-0757-456D-857F-FF12EEEA92AF}'
2 $MsiExec = ${env:SystemRoot} + '\System32\msiexec.exe'
3 $Params = '/x ' + $AppGuid + ' /qn /norestart'
4
5 Start-Process -FilePath $MsiExec -ArgumentList $Params -ErrorVariable +err -Verb Open -Wait
```

```
PS C:\> $AppGuid = '{0F23C126-0757-456D-857F-FF12EEEA92AF}'
$MsiExec = ${env:SystemRoot} + '\System32\msiexec.exe'
$Params = '/x ' + $AppGuid + ' /qn /norestart'

PS C:\> write-host $MsiExec $Params
C:\Windows\System32\msiexec.exe /x {0F23C126-0757-456D-857F-FF12EEEA92AF} /qn /norestart

PS C:\>
```

The status bar at the bottom right indicates the current position is 'Ln 9 Col 9' and the zoom level is '100%'.

Uninstall Methods

MsiExec.exe

Used with an application's GUID in the following command format for SCCM silent uninstalls:

```
msiexec /x {SomeAppGUID} /qn /norestart
```

An **UninstallString** in the format of `msiexec /x{SomeApp-GUID}` can be retrieved using the `(Get-ItemProperty $RegKeyPath).UninstallString` command passed into construction of a parameter string with the correct `msiexec` switches added at the string end.

UninstallString

```
MsiExec.exe /X{1F1C2DFC-2D24-3E06-BCB8-725134ADF989}
```

PowerShell Program

```
# Variables - Modify
$AppName1 = 'Microsoft Visual C++ 2008 Redistributable - x86 9.0.30729.4148'
$AppName1Regkey = 'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{1F1C2DFC-2D24-3E06-BCB8-725134ADF989}'

# Variables - Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'
$StartLocation = Get-Location

# Uninstall AppName1
set-location $StartLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/c "' + (Get-ItemProperty $AppName1Regkey).UninstallString + '
    /qn /norestart"' # Or /qb

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -
    Verb Open -wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}
}
```

An **UninstallString** in the format of `msiexec /i{SomeApp-GUID}`, where `/i` is an interactive switch, is not suited for quiet SCCM uninstalls and will need a parameter string constructed using an `$AppGuid` string variable set to the application GUID with the correct `msiexec` switches added at the beginning and end of the parameter string as in the example below.

UninstallString

```
MsiExec.exe /I{1F1C2DFC-2D24-3E06-BCB8-725134ADF989}
```

PowerShell Program

```
# Variables - Modify
$AppName1 = 'Orca'
$AppName1Guid = '{1F1C2DFC-2D24-3E06-BCB8-725134ADF989}'
$AppName1Regkey =
    'HKLM:SOFTWARE\Wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{039694F1
    -2108-4B3E-8575-85C245210F94}'

# Variables - Do NOT modify
$MsiExec = ${env:SystemRoot} + '\system32\msiexec.exe'
$StartLocation = Get-Location

# Uninstall AppName1
set-location $StartLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $MsiexecParams = '/x ' + $AppName1Guid + ' /qn /norestart' # Or /qb

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $MsiExec $MsiexecParams"
    Write-Host ""
    Start-Process -FilePath $MsiExec -ArgumentList $MsiexecParams -ErrorVariable
        +err -Verb Open -wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}
}
```

Cmd.exe

Used with an application's **UninstallString** or **QuietUninstallString** retrieved from the application's uninstall registry key if the double quote format is correct.

```
cmd /c "path\uninstall.exe" <uninstall_parameters>
```

where the uninstaller executable is contained within the double quotes and any uninstall switches are at the end of the string after the last double quote. The /c switch carries out the command specified by the string and then terminates.

Case 1

An **UninstallString** in the format of "path/uninstaller.exe" switches can be retrieved using the (Get-ItemProperty \$RegKeyPath).UninstallString command passed directly into the cmd uninstall method without any modifications due to the presence of the executable already wrapped correctly in double quotes.

UninstallString

```
"C:\Program Files (x86)\InstallShield Installation Information\{311AD3AF-EA0A-419A-8564-C72442487A0C}\setup.exe" -runfromtemp -l0x0409 -removeonly
```

PowerShell Program

```
# Variables - Modify
$AppName1 = 'DocIT Ls Analysis'
$AppName1Regkey =
    'HKLM:SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\Install
    shield_{311AD3AF-EA0A-419A-8564-C72442487A0C}'

# Variables - Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'
$StartLocation = Get-Location

# Uninstall AppName1
set-location $StartLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/c ' + (Get-ItemProperty $AppName1Regkey).UninstallString

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command:  $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -
        Verb Open -Wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}
}
```

Case 2

The same method above also works for an application's **QuietUninstallString** with the correct format.

QuietUninstallString

```
"C:\Program Files (x86)\SnapGene\Uninstall.exe" /S
```

PowerShell Program

```
# Variables - Modify
$AppName1 = 'SnapGene v2.5'
$AppName1Regkey =
    'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\SnapGene'

# Variables - Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'
$StartLocation = Get-Location

# Uninstall AppName1
set-location $StartLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/c ' + (Get-ItemProperty $AppName1Regkey).QuietUninstallString

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -
        Verb Open -Wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}
}
```

Case 3

An **UninstallString** in the format of path/uninstaller.exe with no switches can be retrieved using the (Get-ItemProperty \$RegKeyPath).UninstallString command passed directly into the cmd uninstall method but the parameter string will need double quotes added around the executable.

UninstallString

c:\totalcmd\tcunin64.exe

PowerShell Program

```
# Variables - Modify
$AppName1 = 'Total Commander 64-bit'
$AppName1Regkey =
    'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Totalcmd64'

# Variables - Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'
$StartLocation = Get-Location

# Uninstall AppName1
set-location $StartLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/c "' + (Get-ItemProperty $AppName1Regkey).UninstallString +
        '"'

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command:  $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -
        Verb Open -Wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}
```

Case 4

Same as **Case 3** above, but with spaces in the application uninstall executable path meaning that double quotes need to be added around the returned **UninstallString** value.

UninstallString

C:\Program Files (x86)\Apperson\CadStd\uninst.exe

PowerShell Program

```
# Variables - Modify
$AppName1 = 'CadStd v3.7'
$AppName1Regkey =
    'HKLM:SOFTWARE\Wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\CadStd'

# Variables - Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'
$StartLocation = Get-Location

# Uninstall AppName1
set-location $StartLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/c "' + (Get-ItemProperty $AppName1Regkey).UninstallString +
        '"'

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -
        Verb Open -Wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}
```


Application Uninstall Executable with No Parameters (Start-Process Method)

An application **UninstallString** in the format of path/uninstaller.exe may use either the **Start-Process** uninstall method with the application's uninstall executable pass into the **-FilePath** parameter or the above method of returning the application's **UninstallString** wrapped in double quotes. If the returned **UninstallString** method does not work, use the **Start-Process** method.

UninstallString

C:\Program Files (x86)\Apperson\CadStd\uninst.exe

PowerShell Program

```
# Variables - Modify
$AppName1 = 'CadStd v3.7'
$AppName1Uninstall = '"' + ${env:ProgramFiles(x86)} +
    '\Apperson\CadStd\uninst.exe"'
$AppName1Regkey =
    'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\CadStd'

# Variables - Do NOT modify
$startLocation = Get-Location

# Uninstall AppName1
set-location $startLocation
write-host "*****"
write-host ""
write-host "Checking for $AppName1 installation..."
write-host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    write-host "    $AppName1 is installed."
    write-host ""
    write-host "Uninstalling $AppName1..."
    write-host ""
    write-host "Command: $AppName1Uninstall"
    write-host ""
    Start-Process -FilePath $AppName1Uninstall -ErrorVariable +err -Verb Open -
        Wait
    write-host ""
    write-host "    $AppName1 uninstall complete."
    write-host ""
}
Else
{
    write-host "    $AppName1 not installed."
    write-host ""
}
```

Application Uninstall Executable with Parameters

An application **UninstallString** in the format of path/uninstaller.exe switches will need to use the following **Start-Process** uninstall method due to double quotes not being around the uninstall executable in the returned registry value. RunDLL32.exe uninstallers are the best example of using this method due to the lack of double quotes around the RunDLL32 executable and double quotes around portions of the uninstall parameter string.

Case 1

A RunDLL32 uninstaller with a complex parameter string:

UninstallString

```
RunDll32
C:\PROGRA~2\COMMON~1\INSTAL~1\PROFES~1\RunTime\09\01\Intel32\Ctor.dll,LaunchSetup
"C:\Program Files (x86)\InstallShield Installation Information\{6D268E5D-4DBA-4B83-9FDA-8655164FDF22}\setup.exe" -l0x9 anything
```

PowerShell Program

```
# Variables - Modify
$AppName1 = 'CadStd v3.7'
$AppName1Uninstall = '"' + ${env:ProgramFiles(x86)} +
    '\Apperson\CadStd\uninst.exe'
$AppName1Params = (${env:SystemDrive} +
    '\PROGRA~2\COMMON~1\INSTAL~1\PROFES~1\RunTime\09\01\Intel32\Ctor.dll,LaunchSetu
p ' + ${env:ProgramFiles(x86)} + '\InstallShield Installation
Information\{6D268E5D-4DBA-4B83-9FDA-8655164FDF22}\setup.exe' -l0x9 anything')
$AppName1Regkey =
    'HKLM:SOFTWARE\Wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{6D268E5D
-4DBA-4B83-9FDA-8655164FDF22}'

# Variables - Do NOT modify
$startLocation = Get-Location

# Uninstall AppName1
set-location $startLocation
write-Host "*****"
write-Host ""
write-Host "Checking for $AppName1 installation..."
write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    write-Host "    $AppName1 is installed."
    write-Host ""
    write-Host "Uninstalling $AppName1..."
    write-Host ""
    write-Host "Command: $AppName1Uninstall $AppName1Params"
    write-Host ""
    Start-Process -FilePath $AppName1Uninstall -ArgumentList $AppName1Params -
        ErrorVariable +err -Verb Open -wait
    write-Host ""
    write-Host "    $AppName1 uninstall complete."
    write-Host ""
}
Else
{
    write-Host "    $AppName1 not installed."
    write-Host ""
}
}
```

Case 2

An uninstaller executable with no double quotes and a parameter string with a path containing spaces:

UninstallString

C:\WINDOWS\unvise32.exe C:\Program Files (x86)\kaleidaGraph 4.0\uninstal.log

PowerShell Program

```
# Variables - Modify
$AppName1 = 'kaleidaGraph 4.0'
$AppName1Uninstall = '%~' + ${env:SystemRoot} + '\unvise32.exe'
$AppName1Params = ${env:ProgramFiles(x86)} + '\kaleidaGraph 4.0\uninstal.log'
$AppName1Regkey =
    'HKLM:SOFTWARE\Wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\kaleidaGr
    aph 4.0'

# Variables - Do NOT modify
$startLocation = Get-Location

# Uninstall AppName1
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""
    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command:  $AppName1Uninstall $AppName1Params"
    Write-Host ""
    Start-Process -FilePath $AppName1Uninstall -ArgumentList $AppName1Params -
        ErrorVariable +err -Verb Open -wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}
}
```

Case 3

An uninstaller executable with no double quotes and a path with spaces followed by a parameter string:

UninstallString

```
C:\Program Files (x86)\Common Files\InstallShield\Driver\8\Intel 32\IDriver.exe  
/M{6D431D78-5A09-47AB-A505-B732663C22F9}
```

PowerShell Program

```
# Variables - Modify  
$AppName1 = 'wallac 1420'  
$AppName1Uninstall = '"' + ${env:ProgramFiles(x86)} + '\Common  
Files\InstallShield\Driver\8\Intel 32\IDriver.exe'"'  
$AppName1Params = '/M{6D431D78-5A09-47AB-A505-B732663C22F9}'  
$AppName1Regkey =  
'HKLM:\SOFTWARE\Wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\InstallSh  
ield_{6D431D78-5A09-47AB-A505-B732663C22F9}'  
  
# Variables - Do NOT modify  
$startLocation = Get-Location  
  
# Uninstall AppName1  
set-location $startLocation  
Write-Host "*****"  
Write-Host ""  
Write-Host "Checking for $AppName1 installation..."  
Write-Host ""  
  
If ((Test-Path -Path $AppName1Regkey) -eq 'True')  
{  
    Write-Host "    $AppName1 is installed."  
    Write-Host ""  
    Write-Host "Uninstalling $AppName1..."  
    Write-Host ""  
    Write-Host "Command: $AppName1Uninstall $AppName1Params"  
    Write-Host ""  
    Start-Process -FilePath $AppName1Uninstall -ArgumentList $AppName1Params -  
        ErrorVariable +err -Verb Open -wait  
    Write-Host ""  
    Write-Host "    $AppName1 uninstall complete."  
    Write-Host ""  
}  
Else  
{  
    Write-Host "    $AppName1 not installed."  
    Write-Host ""  
}
```

EXE - No Parameters and No User Prompt

EXE Uninstaller Example - No Parameters

```
# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Uninstall = ${env:ProgramFiles(x86)} + 'SomeApplication\Uninstall.exe'
$AppName1Regkey =
    'HKLM:\SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errorfileLocation = (${env:SystemDrive} + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Uninstaller"
Write-Host ""
Write-Host "Purpose: Uninstall the following components:"
Write-Host "        - Uninstall $AppName1"
Write-Host ""

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""
    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $AppName1Uninstall"
    Start-Process -FilePath $AppName1Uninstall -ErrorVariable +err -Verb Open -wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 is not installed."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errorfileLocation
Write-Host "Log file location = $errorfileLocation"
Write-Host ""

timeout 5
```

EXE - No Parameters with User Prompt

EXE Uninstaller Example with User Prompt, No Parameters

```
# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Uninstall = ${env:ProgramFiles(x86)} + 'SomeApplication\Uninstall.exe'
$AppName1Regkey =
    'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Uninstaller"
Write-Host ""
Write-Host "Purpose: Remove the following components:"
Write-Host "    - Prompt to uninstall $AppName1"
Write-Host ""

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""
    $title = "$AppName1 Uninstall"
    $message = "Do you want to uninstall $AppName1 ?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
        "Uninstall $AppName1"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
        $AppName1"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($yes, $no)
    $result = $host.ui.PromptForChoice($title, $message, $options, 0)
    Write-Host ""

    Switch ($result)
    {
        0 {"    You selected Yes."}
        1 {"    You selected No."}
    }

    If ($result -eq 0)
    {
        Write-Host ""
        Write-Host "Uninstalling $AppName1..."
        Write-Host ""
        Write-Host "Command:  $AppName1Uninstall"
        Write-Host ""
        Start-Process -FilePath $AppName1Uninstall -ErrorVariable +err -Verb Open -
            wait
        Write-Host ""
        Write-Host "    $AppName1 uninstall complete."
        Write-Host ""
    }
}
Else
{
    Write-Host ""
    Write-Host "    Skipping $AppName1 uninstall."
    Write-Host ""
}
```

```
    }  
  }  
Else  
{  
    Write-Host "    $AppName1 not installed."  
    Write-Host ""  
}  
  
sleep 5  
  
# Write error file  
$err | Out-File $errFileLocation  
Write-Host "*** Log file location = $errFileLocation ***"  
Write-Host ""  
  
timeout 5
```

EXE - Parameters with No User Prompt

EXE Uninstaller Example with Parameters

```
# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Uninstall = ${env:ProgramFiles(x86)} + 'SomeApplication\Uninstall.exe'
$AppName1Params = '/SILENT'
$AppName1Regkey = 'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Uninstaller"
Write-Host ""
Write-Host "Purpose: Uninstall the following components:"
Write-Host "    - Uninstall $AppName1"
Write-Host ""

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""
    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $AppName1Uninstall $AppName1Params"
    Start-Process -FilePath $AppName1Uninstall -ArgumentList $AppName1Params -
        ErrorVariable +err -Verb Open -Wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 is not installed."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```


EXE - Parameters with User Prompt

```
# EXE Uninstaller Example with User Prompt, Parameters

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Uninstall = ${env:ProgramFiles(x86)} + 'SomeApplication\Uninstall.exe'
$AppName1Params = '/SILENT'
$AppName1Regkey =
    'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Uninstaller"
Write-Host ""
Write-Host "Purpose: Remove the following components:"
Write-Host "    - Prompt to uninstall $AppName1"
Write-Host ""

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""
    $title = "$AppName1 Uninstall"
    $message = "Do you want to uninstall $AppName1 ?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
        "Uninstall $AppName1"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
        $AppName1"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($yes, $no)
    $result = $host.ui.PromptForChoice($title, $message, $options, 0)
    Write-Host ""

    Switch ($result)
    {
        0 {"    You selected Yes."}

        1 {"    You selected No."}
    }

    If ($result -eq 0)
    {
        Write-Host ""
        Write-Host "Uninstalling $AppName1..."
        Write-Host ""
        Write-Host "Command: $AppName1Uninstall $AppName1Params"
        Write-Host ""
        Start-Process -FilePath $AppName1Uninstall -ArgumentList $AppName1Params -
            ErrorVariable +err -Verb Open -Wait
        Write-Host ""
        Write-Host "    $AppName1 uninstall complete."
        Write-Host ""
    }
}
Else
{
    Write-Host ""
    Write-Host "    Skipping $AppName1 uninstall."
}
```

```
        Write-Host ""
    }
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5
```

EXE - QuietUninstallString Registry Key

```
# EXE Uninstaller using QuietUninstallString registry key

# Watch out for the double quotes!
# Extra switches should go outside the last double quote preceded by a space.
# Add double quotes if the UninstallString does not have double quotes.

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey = 'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Variables - Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'

# Error file
$StartLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Uninstall Some Application
set-location $StartLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/c ' + (Get-ItemProperty $AppName1Regkey).QuietUninstallString

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -verb
        Open -wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}

sleep 5
```

EXE - UninstallString Registry Key

```
# EXE Uninstaller using UninstallString registry key

# Watch out for the double quotes!
# Extra switches should go outside the last double quote preceded by a space.
# Add double quotes if the UninstallString does not have double quotes.

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey = 'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Variables - Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\ ' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/c ' + (Get-ItemProperty $AppName1Regkey).UninstallString

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -verb
        Open -wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}

sleep 5
```

EXE - UninstallString Registry Key and Double Quotes

```
# EXE Uninstaller using UninstallString registry key

# Watch out for the double quotes!
# Extra switches should go outside the last double quote preceded by a space.
# Add double quotes if the UninstallString does not have double quotes.

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey = 'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Variables - Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\ ' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/c "' + (Get-ItemProperty $AppName1Regkey).UninstallString + '"'

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -verb
        Open -wait
    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 not installed."
    Write-Host ""
}

sleep 5
```

MSI - No User Prompt

```
# MSI Uninstaller Example, No User Prompt

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Guid = '{SomeAppGUID}'
$AppName1Regkey = 'HKLM:\SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Variables -- Do NOT modify
$MsiExec = ('${env:SystemRoot}' + '\System32\msiexec.exe')
$AppName1Params = '/x ' + $AppName1Guid + ' /qb/ norestart' # or /qn switch

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errorFileLocation = ('${env:SystemDrive}' + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Uninstaller"
Write-Host ""
Write-Host "Purpose: Remove the following components:"
Write-Host "        - Uninstall $AppName1"
Write-Host ""

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "        $AppName1 is installed."
    Write-Host ""
    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $MsiExec $AppName1Params"
    Write-Host ""
    Start-Process -FilePath $MsiExec -ArgumentList $AppName1Params -ErrorVariable +err
    -Verb Open -Wait
    Write-Host ""
    Write-Host "        $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "        $AppName1 not installed."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errorFileLocation
Write-Host "Log file location = $errorFileLocation"
Write-Host ""

timeout 5
```

MSI - User Prompt

```
# MSI Uninstaller Example, User Prompt

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Guid = '{SomeAppGUID}'
$AppName1Regkey = 'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Variables -- Do NOT modify
$MsiExec = ('{env:SystemRoot}' + '\System32\msiexec.exe')
$AppName1Params = '/x ' + $AppName1Guid + ' /qb/ norestart' # or /qn switch

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = ('{env:SystemDrive}' + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Uninstaller"
Write-Host ""
Write-Host "Purpose: Remove the following components:"
Write-Host "        - Prompt to uninstall $AppName1"
Write-Host ""

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "        $AppName1 is installed."
    Write-Host ""
    $title = "$AppName1 Uninstall"
    $message = "Do you want to uninstall $AppName1 ?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
        "Uninstall $AppName1"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
        $AppName1"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($yes, $no)
    $result = $host.ui.PromptForChoice($title, $message, $options, 0)
    Write-Host ""

    Switch ($result)
    {
        0 {"        You selected Yes."}
        1 {"        You selected No."}
    }

    If ($result -eq 0)
    {
        Write-Host ""
        Write-Host "Uninstalling $AppName1..."
        Write-Host ""
        Write-Host "Command: $MsiExec $AppName1Params"
        Write-Host ""
        Start-Process -FilePath $MsiExec -ArgumentList $AppName1Params -ErrorVariable
            +err -Verb Open -Wait
        Write-Host ""
        Write-Host "        $AppName1 uninstall complete."
        Write-Host ""
    }
}
Else
```

```
    {
        Write-Host ""
        Write-Host "      Skipping $AppName1 uninstall."
        Write-Host ""
    }
}
Else
{
    Write-Host "      $AppName1 not installed."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5
```


MSI - UninstallString Registry Key, Double Quotes, and Switches

```
# MSI Uninstaller using UninstallString registry key, adding quotes and switches

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Guid = '{SomeAppGUID}'
$AppName1Regkey = 'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeAppGUID}'

# Variables -- Do NOT modify
$Cmd = ${env:SystemRoot} + '\System32\cmd.exe'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Uninstaller"
Write-Host ""
Write-Host "Purpose: Remove the following components:"
Write-Host "      - Uninstall $AppName1"
Write-Host ""

# Uninstall Some Application
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "      $AppName1 is installed."
    Write-Host ""

    $CmdParams = '/x "' + (Get-ItemProperty $AppName1Regkey).UninstallString + '" /qn /norestart' # or /qb switch

    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $Cmd $CmdParams"
    Write-Host ""
    Start-Process -FilePath $Cmd -ArgumentList $CmdParams -ErrorVariable +err -Verb
    Open -wait
    Write-Host ""
    Write-Host "      $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "      $AppName1 not installed."
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5
```

Other Methods

Adding a Path to the PATH Environmental Variable

```
# Adding an application's path to the windows PATH environment variable

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Path = ${env:ProgramFiles(x86)} + '\AppName1'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Add the folder path to the PATH environment variable
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1Path in PATH environment variable..."
Write-Host ""

$oldPath = (Get-ItemProperty -Path
'Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\Environment' -Name PATH).Path

If ($ENV:PATH | Select-String -SimpleMatch $AppName1Path)
{
    Write-Host "    $AppName1Path path already added to PATH environment variable."
    Write-Host ""
}
Else
{
    $NewPath = $oldPath + ';' + $AppName1Path + ';'

    Write-Host "    $AppName1Path path not added to PATH environment variable."
    Write-Host ""
    Write-Host "Adding $AppName1Path path to PATH environment variable..."
    Write-Host ""
    Write-Host "Command: Set-ItemProperty -Path
'Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\Environment' -Name PATH -Value $NewPath"
    Set-ItemProperty -Path
'Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\Environment' -Name PATH -Value $NewPath
    Write-Host ""
    Write-Host "    Adding $AppName1Path path to PATH environment variable complete."
    Write-Host ""
    Return $NewPath
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

Adjusting Token Privileges

This PowerShell snippet comes from an article, *Adjusting Token Privilege with PowerShell*, by Precision Computing (copyright 2010). Retrieved from:

<http://www.leeholmes.com/blog/2010/09/24/adjusting-token-privileges-in-powershell/>

Add the *Adjusting Token Privileges with PowerShell* code to enable the **SeTakeOwnershipPrivilege** token when needing to modify ACLs on system files and folders. Include the commented credit to Precision Computing.

For installing SQL Server, if blocked by domain Group Policy, add the *Adjusting Token Privileges* code to enable the **SeDebugPrivilege** token. Again, include the commented credit to Precision Computing. Some software applications include a licensed version of SQL Server that installs with the application. Enabling the **SeDebugPrivilege** overrides the domain group policy preventing SQL Server installations, thus allowing the software installation to succeed.

This code snippet works for both installation and uninstall programs. Place the snippet at the beginning of your program.

SeTakeOwnershipPrivilege

```
# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs without errors      #
# Privilege token only good for current PowerShell session                      #
#                                                                              #
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2010 #
# http://www.leeholmes.com/blog/2010/09/24/adjusting-token-privileges-in-powershell/ #

function enable-privilege {
    param(
        ## The privilege to adjust. This set is taken from
        ## http://msdn.microsoft.com/en-us/library/bb530716(vs.85).aspx
        [ValidateSet(
            "SeAssignPrimaryTokenPrivilege", "SeAuditPrivilege", "SeBackupPrivilege",
            "SeChangeNotifyPrivilege", "SeCreateGlobalPrivilege", "SeCreatePagefilePrivilege",
            "SeCreatePermanentPrivilege", "SeCreateSymbolicLinkPrivilege",
            "SeCreateTokenPrivilege",
            "SeDebugPrivilege", "SeEnableDelegationPrivilege", "SeImpersonatePrivilege",
            "SeIncreaseBasePriorityPrivilege",
            "SeIncreaseQuotaPrivilege", "SeIncreaseWorkingSetPrivilege",
            "SeLoadDriverPrivilege",
            "SeLockMemoryPrivilege", "SeMachineAccountPrivilege", "SeManageVolumePrivilege",
            "SeProfileSingleProcessPrivilege", "SeRelabelPrivilege",
            "SeRemoteShutdownPrivilege",
            "SeRestorePrivilege", "SeSecurityPrivilege", "SeShutdownPrivilege",
            "SeSyncAgentPrivilege",
            "SeSystemEnvironmentPrivilege", "SeSystemProfilePrivilege",
            "SeSystemtimePrivilege",
            "SeTakeOwnershipPrivilege", "SeTcbPrivilege", "SeTimeZonePrivilege",
            "SeTrustedCredManAccessPrivilege",
            "SeUndockPrivilege", "SeUnsolicitedInputPrivilege")]
        $Privilege,
        ## The process on which to adjust the privilege. Defaults to the current process.
        $ProcessId = $pid,
        ## Switch to disable the privilege, rather than enable it.
        [Switch] $Disable
    )

    ## Taken from P/Invoke.NET with minor adjustments.
    $definition = @"
using System;
using System.Runtime.InteropServices;

public class AdjPriv
```

```

{
[DllImport("advapi32.dll", ExactSpelling = true, SetLastError = true)]
internal static extern bool AdjustTokenPrivileges(IntPtr htok, bool disall,
    ref TokPriv1Luid newst, int len, IntPtr prev, IntPtr relen);

[DllImport("advapi32.dll", ExactSpelling = true, SetLastError = true)]
internal static extern bool OpenProcessToken(IntPtr h, int acc, ref IntPtr phtok);
[DllImport("advapi32.dll", SetLastError = true)]
internal static extern bool LookupPrivilegeValue(string host, string name, ref long
    pluid);
[StructLayout(LayoutKind.Sequential, Pack = 1)]
internal struct TokPriv1Luid
{
    public int Count;
    public long Luid;
    public int Attr;
}

internal const int SE_PRIVILEGE_ENABLED = 0x00000002;
internal const int SE_PRIVILEGE_DISABLED = 0x00000000;
internal const int TOKEN_QUERY = 0x00000008;
internal const int TOKEN_ADJUST_PRIVILEGES = 0x00000020;
public static bool EnablePrivilege(long processHandle, string privilege, bool
    disable)
{
    bool retVal;
    TokPriv1Luid tp;
    IntPtr hproc = new IntPtr(processHandle);
    IntPtr htok = IntPtr.Zero;
    retVal = OpenProcessToken(hproc, TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, ref htok);
    tp.Count = 1;
    tp.Luid = 0;
    if(disable)
    {
        tp.Attr = SE_PRIVILEGE_DISABLED;
    }
    else
    {
        tp.Attr = SE_PRIVILEGE_ENABLED;
    }
    retVal = LookupPrivilegeValue(null, privilege, ref tp.Luid);
    retVal = AdjustTokenPrivileges(htok, false, ref tp, 0, IntPtr.Zero, IntPtr.Zero);
    return retVal;
}
}
'@

$processHandle = (Get-Process -id $ProcessId).Handle
$type = Add-Type $definition -PassThru
$type[0]::EnablePrivilege($processHandle, $Privilege, $Disable)
}

# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs without errors      #
# Privilege token only good for current PowerShell session                        #

enable-privilege SeTakeOwnershipPrivilege |out-null

```

SeDebugPrivilege

```
# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs without errors      #
# Privilege token only good for current PowerShell session                        #
#                                                                              #
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2010 #
# http://www.leeholmes.com/blog/2010/09/24/adjusting-token-privileges-in-powershell/ #

function enable-privilege {
    param(
        ## The privilege to adjust. This set is taken from
        ## http://msdn.microsoft.com/en-us/library/bb530716\(vs.85\).aspx
        [ValidateSet(
            "SeAssignPrimaryTokenPrivilege", "SeAuditPrivilege", "SeBackupPrivilege",
            "SeChangeNotifyPrivilege", "SeCreateGlobalPrivilege", "SeCreatePagefilePrivilege",
            "SeCreatePermanentPrivilege", "SeCreatesymbolicLinkPrivilege",
            "SeCreateTokenPrivilege",
            "SeDebugPrivilege", "SeEnableDelegationPrivilege", "SeImpersonatePrivilege",
            "SeIncreaseBasePriorityPrivilege",
            "SeIncreaseQuotaPrivilege", "SeIncreaseWorkingSetPrivilege",
            "SeLoadDriverPrivilege",
            "SeLockMemoryPrivilege", "SeMachineAccountPrivilege", "SeManageVolumePrivilege",
            "SeProfileSingleProcessPrivilege", "SeRelabelPrivilege",
            "SeRemoteShutdownPrivilege",
            "SeRestorePrivilege", "SeSecurityPrivilege", "SeShutdownPrivilege",
            "SeSyncAgentPrivilege",
            "SeSystemEnvironmentPrivilege", "SeSystemProfilePrivilege",
            "SeSystemtimePrivilege",
            "SeTakeOwnershipPrivilege", "SeTcbPrivilege", "SeTimeZonePrivilege",
            "SeTrustedCredManAccessPrivilege",
            "SeUndockPrivilege", "SeUnsolicitedInputPrivilege")]
        $Privilege,
        ## The process on which to adjust the privilege. Defaults to the current process.
        $ProcessId = $pid,
        ## Switch to disable the privilege, rather than enable it.
        [Switch] $Disable
    )

    ## Taken from P/Invoke.NET with minor adjustments.
    $definition = @"
using System;
using System.Runtime.InteropServices;

public class AdjPriv
{
    [DllImport("advapi32.dll", ExactSpelling = true, SetLastError = true)]
    internal static extern bool AdjustTokenPrivileges(IntPtr htok, bool disall,
        ref TokPriv1Luid newst, int len, IntPtr prev, IntPtr relen);

    [DllImport("advapi32.dll", ExactSpelling = true, SetLastError = true)]
    internal static extern bool OpenProcessToken(IntPtr h, int acc, ref IntPtr phtok);
    [DllImport("advapi32.dll", SetLastError = true)]
    internal static extern bool LookupPrivilegeValue(string host, string name, ref long pluid);
    [StructLayout(LayoutKind.Sequential, Pack = 1)]
    internal struct TokPriv1Luid
    {
        public int Count;
        public long Luid;
        public int Attr;
    }

    internal const int SE_PRIVILEGE_ENABLED = 0x00000002;
    internal const int SE_PRIVILEGE_DISABLED = 0x00000000;
    internal const int TOKEN_QUERY = 0x00000008;
    internal const int TOKEN_ADJUST_PRIVILEGES = 0x00000020;
    public static bool EnablePrivilege(long processHandle, string privilege, bool disable)
    {
        bool retVal;
        TokPriv1Luid tp;
```

```

IntPtr hproc = new IntPtr(processHandle);
IntPtr htok = IntPtr.Zero;
retVal = OpenProcessToken(hproc, TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, ref htok);
tp.Count = 1;
tp.Luid = 0;
if(disable)
{
    tp.Attr = SE_PRIVILEGE_DISABLED;
}
else
{
    tp.Attr = SE_PRIVILEGE_ENABLED;
}
retVal = LookupPrivilegeValue(null, privilege, ref tp.Luid);
retVal = AdjustTokenPrivileges(htok, false, ref tp, 0, IntPtr.Zero, IntPtr.Zero);
return retVal;
}
}
'@

```

```

$processHandle = (Get-Process -id $ProcessId).Handle
$type = Add-Type $definition -PassThru
$type[0]::EnablePrivilege($processHandle, $Privilege, $Disable)
}

```

```

# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs without errors      #
# Privilege token only good for current PowerShell session                        #
enable-privilege SeDebugPrivilege |out-null

```

Compatibility Mode

When installing older software, the compatibility mode may need to be set on the installer executable, the application executable, and/or the uninstaller executable. Any compatibility mode settings that are created during an install also need to be removed during an uninstall. However, if there is more than one application using an installer named setup (or uninstall) on the local system, this step should be skipped to avoid removing a compatibility mode setting which may impact another application's functionality.

Installer

```
# Setting Compatibility Mode on the install and uninstall executables

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Install = ('' + $startLocation + '\setup.exe')
$AppName1Uninstall = ${env:ProgramFiles(x86)} + '\AppName1\uninstall.exe'
$AppName1Regkey =
    'HKLM:SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeApp-
    GUID}'
$CompMode = 'WINXPSP2 RUNASADMIN'
$InstallExe = 'setup.exe'
$UninstallExe = 'uninstall.exe'

# Variables -- Do NOT modify
$CompModePath =
    'HKLM:SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Installer"
Write-Host ""
Write-Host "Purpose: Install the following components:"
Write-Host "    - Set $InstallExe compatibility mode to $CompMode"
Write-Host "    - Install $AppName1"
Write-Host "    - Set $UninstallExe compatibility mode to $CompMode"
Write-Host ""

# Set $InstallExe compatibility mode to $CompMode
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installer compatibility mode setting..."
Write-Host ""

if ((Get-ItemProperty $CompModePath) -match $InstallExe)
{
    Write-Host "    $AppName1 installer compatibility mode setting already exists."
    Write-Host ""
}
else
{
    Write-Host "Setting $AppName1 installer compatibility mode to $CompMode..."
    Write-Host ""
    Write-Host "Command: New-ItemProperty -Path $CompModePath -name $AppName1Install -"
    Write-Host "    Value $CompMode"
    Write-Host ""

    New-ItemProperty -Path $CompModePath -Name $AppName1Install -Value $CompMode

    Write-Host ""
```

```

        Write-Host "    $AppName1 installer compatibility mode setting complete."
        Write-Host ""
    }

    sleep 5

    # Install $AppName1
    set-location $startLocation
    Write-Host "*****"
    Write-Host ""
    Write-Host "Checking for $AppName1 installation..."
    Write-Host ""

    # Check uninstall registry key or install path
    If ((Test-Path -Path $AppName1Regkey) -ne 'True')
    {
        Write-Host "    $AppName1 is not installed."
        Write-Host ""
        Write-Host "Installing $AppName1..."
        Write-Host ""
        Write-Host "Command: $AppName1Install"
        Write-Host ""

        Start-Process -FilePath $AppName1Install -ErrorVariable +err -Verb Open -Wait

        Write-Host ""
        Write-Host "    $AppName1 install complete."
        Write-Host ""
    }
    Else
    {
        Write-Host "    $AppName1 is already installed."
        Write-Host ""
    }

    sleep 5

    # Set $UninstallExe compatibility mode to $CompMode
    set-location $startLocation
    Write-Host "*****"
    Write-Host ""
    Write-Host "Checking for $AppName1 uninstaller compatibility mode setting..."
    Write-Host ""

    if ((Get-ItemProperty $CompModePath) -match $UninstallExe)
    {
        Write-Host "    $AppName1 uninstaller compatibility mode setting already exists."
        Write-Host ""
    }
    else
    {
        Write-Host "Setting $AppName1 uninstaller compatibility mode to $CompMode..."
        Write-Host ""
        Write-Host "Command: New-ItemProperty -Path $CompModePath -name $AppName1Uninstall"
        Write-Host "    -Value $CompMode"
        Write-Host ""

        New-ItemProperty -Path $CompModePath -Name $AppName1Uninstall -Value $CompMode

        Write-Host ""
        Write-Host "    $AppName1 uninstaller compatibility mode setting complete."
        Write-Host ""
    }

    sleep 5

    # Write error file
    $err | Out-File $errFileLocation
    Write-Host "**** Log file location = " $errFileLocation " ****"
    Write-Host ""

    timeout 5

```


Uninstaller

```
# Removing Compatibility Mode settings on the install and uninstall executables

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Uninstall = ${env:ProgramFiles(x86)} + '\AppName1\uninstall.exe'
$AppName1Params = '/SILENT'
$AppName1Regkey =
    'HKLM:SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeApp-
    GUID}'
$AppName1InstallFolder = (${env:SystemDrive} + '\AppName1')

# Remove WINXPSP2 compatibility settings -- Modify per individual application
$AppName1InstallValue = 'setup.exe'
$AppName1UninstallValue = 'uninstall.exe'
$AppName1InstallSetting = '*setup.exe'
$AppName1UninstallSetting = '*uninstall.exe'

# Variables -- Do NOT modify
$CompModePath = 'HKLM:SOFTWARE\Microsoft\windows
    NT\CurrentVersion\AppCompatFlags\Layers'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

Write-Host ""
Write-Host "*****"
Write-Host ""
Write-Host "$AppName1 Uninstaller"
Write-Host ""
Write-Host "Purpose: Remove the following components:"
Write-Host "    - Uninstall $AppName1"
Write-Host "    - Remove $AppName1 compatibility settings"
Write-Host ""

# Uninstall AppName1 with parameters
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

# Check uninstall registry key or install path
If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""
    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $AppName1Uninstall $AppName1Params"
    Write-Host ""

    Start-Process -FilePath $AppName1Uninstall -ArgumentList $AppName1Params -
        ErrorVariable +err -Verb Open -wait

    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 is not installed."
    Write-Host ""
}

sleep 5
```

```

# Delete leftover install folder
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 install folder..."
Write-Host ""

if ((Test-Path -Path $AppName1InstallFolder) -eq 'True')
{
    Write-Host "    $AppName1 install folder exists."
    Write-Host ""
    Write-Host "Deleting $AppName1 install folder..."
    Write-Host ""
    Write-Host "Command: Remove-Item -Path $AppName1InstallFolder -Recurse -Force"
    Write-Host ""

    Remove-Item -Path $AppName1InstallFolder -Recurse -Force

    Write-Host ""
    Write-Host "    $AppName1 install folder deletion complete."
    Write-Host ""
}
else
{
    Write-Host "    $AppName1 install folder does not exist."
    Write-Host ""
}

sleep 5

# Remove WINXPSP2 compatibility settings on installer executable
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installer compatibility setting..."
Write-Host ""

if ((Get-ItemProperty $CompModePath) -match $AppName1InstallValue)
{
    Write-Host "Deleting $AppName1 installer compatibility setting..."
    Write-Host ""
    Write-Host "Command: Remove-ItemProperty -Path $CompModePath -name $AppName1InstallSetting -Force"
    Write-Host ""

    Remove-ItemProperty -Path $CompModePath -name $AppName1InstallSetting -Force
}
else
{
    Write-Host "    $AppName1 installer compatibility setting does not exist."
    Write-Host ""
}

sleep 5

# Remove WINXPSP2 compatibility settings on uninstaller executable
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 uninstaller compatibility mode setting..."
Write-Host ""

if ((Get-ItemProperty $CompModePath) -match $AppName1UninstallValue)
{
    Write-Host "Deleting $AppName1 uninstaller compatibility setting..."
    Write-Host ""
    Write-Host "Command: Remove-ItemProperty -Path $CompModePath -name $AppName1UninstallSetting -Force"
    Write-Host ""

    Remove-ItemProperty -Path $CompModePath -name $AppName1UninstallSetting -Force
}
else
{
    Write-Host "    $AppName1 uninstaller compatibility mode setting does not exist."
    Write-Host ""
}

sleep 5

```

```
}  
else  
{  
    Write-Host "" $AppName1 uninstaller compatibility setting does not exist."  
    Write-Host ""  
}  
  
Sleep 5  
  
# write error file  
$err | Out-File $errFileLocation  
Write-Host "*** Log file location = " $errFileLocation " ***"  
Write-Host ""  
  
timeout 5
```

Determining an Application's Install Location

The following snippet finds an application's install location.

```
# Determining an application's install location

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey = 'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{SomeApp-GUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\ ' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Verify AppName1 is installed
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

# Check uninstall registry key or install path
If ((Test-Path -Path $AppName1Regkey) -ne 'True')
{
    Write-Host "    $AppName1 is not installed!"
    Write-Host ""
    Write-Host "Please install $AppName1 before continuing..."
    Write-Host ""
    Write-Host "    Press any key to quit..."
    Pause
    Break    # Halts program execution
}
Else
{
    Write-Host "    $AppName1 already installed."
    Write-Host ""

    # Get AppName1 InstallLocation from registry
    $AppName1Location = (Get-ItemProperty -Path $AppName1Regkey -Name
        InstallLocation).InstallLocation

    Write-Host "$AppName1 located at: $AppName1Location"
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

Enable "Show hidden files, etc" in Folder Options

This snippet enables the **Show hidden files, folders, etc.** option in Windows Explorer **Folder Options** by changing the relevant registry key value.

```
# Enabling 'Show hidden files, etc' option in Windows Folder Options

# Variable -- Modify
$AppName1 = 'Some Application'

# Variable -- Do NOT modify
$ExplorerHiddenRegkey =
    'HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\ ' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = ($env:SystemDrive + $errorpath)

# Enable option in registry
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Enabling 'Show hidden files, etc' in Folder Options..."
Write-Host ""
Write-Host "Command: Set-ItemProperty $ExplorerHiddenRegkey Hidden 1"
Write-Host ""
Set-ItemProperty $ExplorerHiddenRegkey Hidden 1

Sleep 2

Write-Host ""
Write-Host "Stopping and restarting the Explorer process..."
Write-Host ""
Write-Host "Command: Stop-Process -processname explorer"
Write-Host ""
Stop-Process -processname explorer
Write-Host ""
Write-Host " 'Show hidden files...' enable complete."
Write-Host ""

Sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

File/Folder ACL Manipulation

Add the *Adjusting Token Privileges with PowerShell* code, including the credit for Precision Computing, at the beginning of a PowerShell install or uninstall program to enable the **SeTakeOwnershipPrivilege** when modifying ACLs on system files and folders. If the files and folders have been installed by SCCM, no ownership will have been assigned as the SYSTEM account cannot own anything. Add PowerShell snippet from the **Granting File/Folder Ownership** section that uses [takeown](#) to assign membership to the Administrator before making any attempts to change file and folder ACLs.

Installer

```
# File/Folder ACL Manipulation - Install

# MODIFY - Application Specific Variables
$AppName1 = 'Some Application'
$Folder1 = (${env:ProgramFiles(x86)} + '\AppName1')

# MODIFY - DOMAIN\domainGroup, BUILTIN\localGroup, etc.
$User = 'BUILTIN\Users'

# MODIFY - FullControl, ReadOnly, write, etc
$FolderRights = 'write'

# MODIFY - FullControl permission does not need 'Synchronize' for string matching
$FolderRightsLong = 'write, Synchronize'

# MODIFY - Set to 'NONE' for files
$Flags = 'ContainerInherit, ObjectInherit'

# MODIFY - Allow, Deny
$AllowDeny = 'Allow'

#####
# DO NOT MODIFY
#
$objUser = New-Object System.Security.Principal.NTAccount("$User")
$colRights = [System.Security.AccessControl.FileSystemRights]$FolderRights
$InheritanceFlag = [System.Security.AccessControl.InheritanceFlags]$Flags
$PropagationFlag = [System.Security.AccessControl.PropagationFlags]::None
$objType = [System.Security.AccessControl.AccessControlType]::$AllowDeny
$objAr = New-Object System.Security.AccessControl.FileSystemAccessRule($objUser,
$colRights, $InheritanceFlag, $PropagationFlag, $objType)
#
# DO NOT MODIFY
#####

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Grant $User group $AllowDeny $FolderRights to $folder1
# - Set ACL on $folder1 folder #
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $User group $AllowDeny $FolderRights permission for $folder1"
Write-Host "folder..."
Write-Host ""

$r = (Get-Acl $folder1).Access | where {$_.IdentityReference -eq $User -and
    $_.FileSystemRights -eq $FolderRightsLong -and $_.AccessControlType -eq
    $AllowDeny}

If ($r -eq $null)
{
```

```

Write-Host "    $User group does not have $AllowDeny $FolderRights permission for
    $folder1 folder."
Write-Host ""

$objAc11 = Get-Acl $folder1
$string = ('' + $objAc11 + '.SetAccessRule(' + $objAr + ')')

Write-Host ""
Write-Host "Granting $User group $AllowDeny $FolderRights permission for $folder1
    folder.."
Write-Host ""
Write-Host "Command: $string"
Write-Host ""

$objAc11.SetAccessRule($objAr)

Write-Host ""
Write-Host "Setting new access list on $folder1 folder..."
Write-Host ""
Write-Host "Command: Set-Acl $folder1 $objAc11"
Write-Host ""

Set-Acl $folder1 $objAc11

Write-Host ""
Write-Host "    New $folder1 access list set complete."
Write-Host ""
}
Else
{
    Write-Host "    $User already has $AllowDeny $FolderRights permission for $folder1
        folder."
    Write-Host ""
}
}

Sleep 5

# write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5

```

Uninstaller

```
# File/Folder ACL Manipulation Example - Uninstall on windows System Folder

# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs without errors
# Privilege token only good for this Powershell session
#
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2010 #
# http://www.leeholmes.com/blog/2010/09/24/adjusting-token-privileges-in-powershell/ #

function enable-privilege {
    param(
        ## The privilege to adjust. This set is taken from
        ## http://msdn.microsoft.com/en-us/library/bb530716\(vs.85\).aspx
        [ValidateSet(
            "SeAssignPrimaryTokenPrivilege", "SeAuditPrivilege", "SeBackupPrivilege",
            "SeChangeNotifyPrivilege", "SeCreateGlobalPrivilege", "SeCreatePagefilePrivilege",
            "SeCreatePermanentPrivilege", "SeCreatesymbolicLinkPrivilege",
            "SeCreateTokenPrivilege",
            "SeDebugPrivilege", "SeEnableDelegationPrivilege", "SeImpersonatePrivilege",
            "SeIncreaseBasePriorityPrivilege",
            "SeIncreaseQuotaPrivilege", "SeIncreaseWorkingSetPrivilege",
            "SeLoadDriverPrivilege",
            "SeLockMemoryPrivilege", "SeMachineAccountPrivilege", "SeManageVolumePrivilege",
            "SeProfileSingleProcessPrivilege", "SeRelabelPrivilege",
            "SeRemoteShutdownPrivilege",
            "SeRestorePrivilege", "SeSecurityPrivilege", "SeShutdownPrivilege",
            "SeSyncAgentPrivilege",
            "SeSystemEnvironmentPrivilege", "SeSystemProfilePrivilege",
            "SeSystemtimePrivilege",
            "SeTakeOwnershipPrivilege", "SeTcbPrivilege", "SeTimeZonePrivilege",
            "SeTrustedCredManAccessPrivilege",
            "SeUndockPrivilege", "SeUnsolicitedInputPrivilege")]
        $Privilege,
        ## The process on which to adjust the privilege. Defaults to the current process.
        $ProcessId = $pid,
        ## Switch to disable the privilege, rather than enable it.
        [Switch] $Disable
    )

    ## Taken from P/Invoke.NET with minor adjustments.
    $definition = @"
using System;
using System.Runtime.InteropServices;

public class AdjPriv
{
    [DllImport("advapi32.dll", ExactSpelling = true, SetLastError = true)]
    internal static extern bool AdjustTokenPrivileges(IntPtr htok, bool disall,
        ref TokPrivtLuid newst, int len, IntPtr prev, IntPtr relen);

    [DllImport("advapi32.dll", ExactSpelling = true, SetLastError = true)]
    internal static extern bool OpenProcessToken(IntPtr h, int acc, ref IntPtr phtok);
    [DllImport("advapi32.dll", SetLastError = true)]
    internal static extern bool LookupPrivilegeValue(string host, string name, ref long
        pluid);
    [StructLayout(LayoutKind.Sequential, Pack = 1)]
    internal struct TokPrivtLuid
    {
        public int Count;
        public long Luid;
        public int Attr;
    }

    internal const int SE_PRIVILEGE_ENABLED = 0x00000002;
    internal const int SE_PRIVILEGE_DISABLED = 0x00000000;
    internal const int TOKEN_QUERY = 0x00000008;
    internal const int TOKEN_ADJUST_PRIVILEGES = 0x00000020;
    public static bool EnablePrivilege(long processHandle, string privilege, bool
        disable)
    {

```



```

    bool retVal;
    TokPriv1Luid tp;
    IntPtr hproc = new IntPtr(processHandle);
    IntPtr htok = IntPtr.Zero;
    retVal = OpenProcessToken(hproc, TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, ref htok);
    tp.Count = 1;
    tp.Luid = 0;
    if(disable)
    {
        tp.Attr = SE_PRIVILEGE_DISABLED;
    }
    else
    {
        tp.Attr = SE_PRIVILEGE_ENABLED;
    }
    retVal = LookupPrivilegeValue(null, privilege, ref tp.Luid);
    retVal = AdjustTokenPrivileges(htok, false, ref tp, 0, IntPtr.Zero, IntPtr.Zero);
    return retVal;
}
}
'@

$processHandle = (Get-Process -id $ProcessId).Handle
$type = Add-Type $definition -PassThru
$type[0]::EnablePrivilege($processHandle, $Privilege, $Disable)
}
enable-privilege SeTakeOwnershipPrivilege |out-null

# MODIFY - Application Specific Variables
$AppName1 = 'Some Application'

# MODIFY - Requires 'SeTakeOwnershipPrivilege' enabled token for system folder ACLs
$folder1 = ($env:windir + '\twain_32')

# MODIFY - DOMAIN\domainGroup, BUILTIN\localGroup, etc.
$User = 'BUILTIN\Users'

# MODIFY - FullControl, ReadOnly, write, etc.
$FolderRights = 'FullControl'

# MODIFY - FullControl does not need 'Synchronize' for string matching
$FolderRightsLong = 'FullControl'

# MODIFY - Set to 'NONE' for file ACLs
$Flags = 'ContainerInherit,ObjectInherit'

# MODIFY - Allow, Deny
$AllowDeny = 'Deny'

#####
# DO NOT MODIFY
#
$objUser = New-Object System.Security.Principal.NTAccount("$User")
$colRights = [system.Security.AccessControl.FileSystemRights]$FolderRights
$InheritanceFlag = [System.Security.AccessControl.InheritanceFlags]$Flags
$PropagationFlag = [System.Security.AccessControl.PropagationFlags]::None
$objType = [system.Security.AccessControl.AccessControlType]::$AllowDeny
$objAr = New-Object System.Security.AccessControl.FileSystemAccessRule($objUser,
$colRights, $InheritanceFlag, $PropagationFlag, $objType)
#
# DO NOT MODIFY
#####

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = ($env:SystemDrive + $errorpath)

# Remove $User group $AllowDeny $FolderRights permissions with user prompt
set-location $startLocation
write-Host "*****"

```

```

Write-Host ""
Write-Host "Checking for $User group $AllowDeny $FolderRights permissions for $folder1
folder..."
Write-Host ""

$r = (Get-Acl $folder1).Access | where {$_.IdentityReference -eq $User -and
    $_.FileSystemRights -eq $FolderRights -and $_.AccessControlType -eq $AllowDeny}

If ($r -ne $null)
{
    Write-Host "    $User group has $AllowDeny $FolderRights permissions for $folder1
    folder"
    Write-Host ""
    $title = "Remove $User Permissions"
    $message = "Do you want to remove the $User group $AllowDeny $FolderRights
    permissions for $folder1 folder?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
    "Remove $User group"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
    $User group"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($yes, $no)
    $result = $host.ui.PromptForChoice($title, $message, $options, 0)
    Write-Host ""

    Switch ($result)
    {
        0 {"    You selected Yes."}
        1 {"    You selected No."}
    }

    If ($result -eq 0)
    {
        $objAcl2 = Get-Acl $folder1
        $string = ('' + $objAcl2 + '.RemoveAccessRule(' + $objAr + ')')

        Write-Host ""
        Write-Host "Removing $User group $AllowDeny $FolderRights permissions for
        $folder1 folder..."
        Write-Host ""
        Write-Host "Command: $string"
        Write-Host ""

        $objAcl2.RemoveAccessRule($objAr)

        Write-Host ""
        Write-Host "Setting new access list on $folder1 folder..."
        Write-Host ""
        Write-Host "Command: Set-Acl $folder1 $objAcl2"
        Write-Host ""

        Set-Acl $folder1 $objAcl2

        Write-Host ""
        Write-Host "    New $folder1 access list set complete."
        Write-Host ""
    }
}
Else
{
    Write-Host ""
    Write-Host "    Skipping $User group $AllowDeny $FolderRights permission check
    on $folder1 folder."
    Write-Host ""
}
}
Else
{
    Write-Host "    $User group does not have $AllowDeny $FolderRights permissions for
    $folder1 folder."
    Write-Host ""
}
}

```

```
sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5
```

File/Folder Copy Methods

If files and/or folders need to be copied during the install, place them in their own separate folder in the Software Vault application folder so that folder will be copied to the local SCCM cache. Then, the PowerShell install program can copy the folder to the proper location on the local system.

Copy-Item Example

```
# File Copy Example Using PowerShell 'Copy-Item'

# Variables -- Modify
$AppName1 = 'Some Application'
$CopyFileName1 = '.\FileName1'
$CopyFileName2 = '.\FileName2'
$FileCopyDestination = ($env:ProgramFiles(x86)} + '\AppName1\')

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = ($env:SystemDrive} + $errorpath)

# Copy two files from the SCCM cache folder to the local drive
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Copying two files to local drive..."
Write-Host ""
Write-Host "Command: Copy-Item $CopyFileName1 $FileCopyDestination -Force"

Copy-Item $CopyFileName1 $FileCopyDestination -Force

Write-Host ""
Write-Host "Command: Copy-Item $CopyFileName2 $FileCopyDestination -Force"

Copy-Item $CopyFileName2 $FileCopyDestination -Force

Write-Host ""
Write-Host "    File copy complete."
Write-Host ""

Sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5
```

xcopy Example

```
# File/Folder Copy Example Using 'xcopy'
#
#     watch out for the two sets of double quotes!
#

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1SourceFolder = '.\SourceFolder\*'
$AppName1DestinationFolder = ${env:ProgramFiles(x86)} + '\TargetFolder\'

# Variables -- Do NOT modify
$CmdPath = (${env:SystemRoot} + '\System32\cmd.exe')
$XcopyString = '/c xcopy "' + $AppName1SourceFolder + '" "' +
$AppName1DestinationFolder + '" /e'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errorFileLocation = (${env:SystemDrive} + $errorpath)

# Copy AppName1 source folder to destination folder
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 source folder on local drive..."
Write-Host ""

If ((Test-Path -Path $AppName1DestinationFolder) -ne 'True')
{
    Write-Host "    $AppName1 source folder on local drive does not exist."
    Write-Host ""
    Write-Host "Copying $AppName1 source folder to local drive..."
    Write-Host ""
    Write-Host "Command:  $CmdPath $XcopyString"

    Start-Process -FilePath $CmdPath -ArgumentList $XcopyString -ErrorVariable +err -
        Verb Open -wait

    Write-Host ""
    Write-Host "    Copying $AppName1 source folder to local drive complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 source folder on local drive already exists."
    Write-Host ""
}
sleep 5

# Write error file
$err | Out-File $errorFileLocation
Write-Host "*** Log file location = $errorFileLocation ***"
Write-Host ""
timeout 5
```

File/Folder Deletion Methods

Sometimes uninstallers leave things behind like install folders, data folders, Start Menu folders, shortcuts, and more.

Delete an Install Folder

```
# Delete a leftover install folder

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Folder = (${env:ProgramFiles} + '\AppName1')

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errorFileLocation = (${env:SystemDrive} + $errorpath)

# Delete leftover $AppName1 install folder
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 install folder..."
Write-Host ""

if ((Test-Path -Path $AppName1Folder) -eq 'True')
{
    Write-Host "    $AppName1 install folder exists."
    Write-Host ""
    Write-Host "Deleting $AppName1 install folder..."
    Write-Host ""
    Write-Host "Command: Remove-Item -Path $AppName1Folder -Recurse -Force"
    Write-Host ""

    Remove-Item -Path $AppName1Folder -Recurse -Force

    Write-Host ""
    Write-Host "    $AppName1 install folder deletion complete."
    Write-Host ""
}
else
{
    Write-Host "    $AppName1 install folder does not exist."
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errorFileLocation
Write-Host "*** Log file location = $errorFileLocation ***"
Write-Host ""

timeout 5
```

Delete a Start Menu Folder

```
# Delete a leftover Start Menu folder

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1StartMenu = ({env:ProgramData} + 'Microsoft\windows\Start
Menu\Programs\AppName1')

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = ({env:SystemDrive} + $errorpath)

# Delete leftover $AppName1 Start Menu folder
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 Start Menu folder..."
Write-Host ""

if ((Test-Path -Path $AppName1StartMenu) -eq 'True')
{
    Write-Host "    $AppName1 Start Menu folder exists."
    Write-Host ""
    Write-Host "Deleting $AppName1 Start Menu folder..."
    Write-Host ""
    Write-Host "Command: Remove-Item -Path $AppName1StartMenu -Recurse -Force"
    Write-Host ""

    Remove-Item -Path $AppName1StartMenu -Recurse -Force

    Write-Host ""
    Write-Host "    $AppName1 Start Menu folder deletion complete."
    Write-Host ""
}
else
{
    Write-Host "    $AppName1 Start Menu folder does not exist."
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5
```

Delete a Desktop Shortcut

```
# Delete a leftover shortcut from the Default Users Desktop

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Shortcut = (${env:SystemDrive} + '\Users\Default\Desktop\AppName1.lnk')

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\ ' + $AppName1 + '_UninstallErrorLog.txt')
$errorFileLocation = (${env:SystemDrive} + $errorpath)

# Delete leftover shortcut from the Default User's Desktop
set-location $startLocation
Write-Host "Checking for $AppName1 shortcut on Default Users Desktop..."
Write-Host ""

if ((Test-Path -Path $AppName1Shortcut) -eq 'True')
{
    Write-Host "    $AppName1 shortcut exists on Default Users Desktop."
    Write-Host ""
    Write-Host "Removing $AppName1 shortcut..."
    Write-Host ""
    Write-Host "Command: Remove-Item -Path $AppName1Shortcut -Force"
    Write-Host ""

    Remove-Item -Path $AppName1Shortcut -Force

    Write-Host ""
    Write-Host "    $AppName1 shortcut removal complete."
    Write-Host ""
}
else
{
    Write-Host "    $AppName1 shortcut does not exist."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errorFileLocation
Write-Host "Log file location = $errorFileLocation"
Write-Host ""

timeout 5
```


Granting File/Folder Ownership

When SCCM creates/copies files and folders to a target system, no owner is assigned to the target files and folders because SCCM runs in the "SYSTEM" context and the "SYSTEM" account cannot own files and folders.

To be able to manipulate the permissions on files and folder using the PowerShell `Set-Acl` command, one must first assign ownership of the object(s) to the Administrators group.

```
# Grant Administrators Ownership Example -- Place before any file/folder ACL
manipulations

# Variables -- Modify
$AppName1 = 'Some Application'
$folder1 = (${env:ProgramFiles(x86)} + '\AppName1')

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errorfileLocation = (${env:SystemDrive} + $errorpath)

# Grant Administrators ownership of $folder1
Write-Host "*****"
Write-Host ""
Write-Host "Granting Ownership of install folder $folder1 to Administrators group..."
Write-Host ""
Write-Host "Command: takeown /F $folder1 /R /A"

takeown /F $folder1 /R /A

Write-Host ""
Write-Host "      Install folder Ownership granting complete."
Write-Host ""

Sleep 5

# Write error file
$err | Out-File $errorfileLocation
Write-Host "*** Log file location = $errorfileLocation ***"
Write-Host ""

timeout 5
```

Group Membership

Some installs have special instructions calling for a domain group to be added to a local user group on the target system. During the uninstall process, any changes in group membership need to be reverted.

Installer

```
# Adding a DomainGroup to a LocalGroup Example -- Install

# Variables -- Modify
$AppName1 = 'Some Application'
$DomainGroup = 'DomainGroup'      # MODIFY - Security, fsgXXXX, etc.
$Domain = 'DOMAIN'                # MODIFY
$LocalGroup = 'LocalGroup'        # MODIFY - Administrators, Users, Power Users, etc

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = ($env:SystemDrive + $errorpath)

# Add $DomainGroup group to $LocalGroup group
set-location $startLocation
write-host "*****"
write-host ""
write-host "Checking for $DomainGroup domain group membership in local $LocalGroup"
write-host "group..."
write-host ""

$r = (([ADSI]"winNT://./$LocalGroup").Invoke("IsMember",
"winNT://$Domain/$DomainGroup"))

If ($r -match 'False')
{
    $string = '[ADSI]"winNT://./' + $LocalGroup + ',group').Add("winNT://" + $Domain +
'/' + $DomainGroup + '"')

    write-host "Adding $DomainGroup domain group to local $LocalGroup group..."
    write-host ""
    write-host "Command: ' $string"
    write-host ""

    ([ADSI]"winNT://./$LocalGroup,group").Add("winNT://$Domain/$DomainGroup")

    write-host ""
    write-host "    Adding $DomainGroup domain group to local $LocalGroup group"
    write-host "complete."
    write-host ""
}
Else
{
    write-host "    $DomainGroup domain group is already a member of local $LocalGroup"
    write-host "group."
    write-host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
write-host "Log file location = $errFileLocation"
write-host ""

timeout 5
```

Uninstaller

```
# Removing a DomainGroup from a LocalGroup Example -- Uninstall with user prompt

# Variables -- Modify
$AppName1 = 'Some Application' # MODIFY
$DomainGroup = 'DomainGroup' # MODIFY - Security, fsgXXXX, etc.
$Domain = 'DOMAIN' # MODIFY
$LocalGroup = 'LocalGroup' # MODIFY - Administrators, Users, Power Users, etc

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Remove $DomainGroup group from $LocalGroup group
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $DomainGroup domain group membership in local $LocalGroup"
Write-Host "group..."
Write-Host ""

$r = (([ADSI]"winNT://./$LocalGroup").Invoke("IsMember",
"winNT://$Domain/$DomainGroup"))

If ($r -match 'True')
{
    Write-Host "    $DomainGroup domain group is member of local $LocalGroup group."
    Write-Host ""
    $title = "$DomainGroup Removal"
    $message = "Do you want to remove the $DomainGroup domain group from the local"
    $LocalGroup "group?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
    "Remove $DomainGroup group"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave"
    $DomainGroup "group"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($yes, $no)
    $result = $host.ui.PromptForChoice($title, $message, $options, 0)
    Write-Host ""

    Switch ($result)
    {
        0 {"    You selected Yes."}
        1 {"    You selected No."}
    }

    If ($result -eq 0)
    {
        $string = '[ADSI]"winNT://./' + $LocalGroup + ',group').Remove("winNT://' +
        $Domain + '/' + $DomainGroup + '"')

        Write-Host ""
        Write-Host "Removing $DomainGroup domain group from local $LocalGroup"
        Write-Host "group..."
        Write-Host ""
        Write-Host "Command: $string"
        Write-Host ""

        ([ADSI]"winNT://./$LocalGroup,group").Remove("winNT://$Domain/$DomainGroup")

        Write-Host ""
        Write-Host "    $DomainGroup domain group removal complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping $DomainGroup domain group removal."
    }
}
```

```
        }
        Write-Host ""
    }
Else
{
    Write-Host "    $DomainGroup domain group is not member of local $LocalGroup
group."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5
```

Registry Key Deletion

Uninstallers very seldom leave behind an **Uninstall** registry key. When it does happen, the **Uninstall** registry key will need to be removed using a PowerShell uninstall program so that SCCM will "discover" that the application is no longer present on the target system.

The easiest way to tell if an Uninstall registry key is still present is by opening the **Programs and Features** control panel applet and refreshing the window. If the application is still listed after the uninstall completes, check the registry for an **Uninstall** key. There may also be an **Installer** registry key that will need to be removed using PowerShell.

```
# Delete a leftover Uninstall registry key

# Variables -- Modify
$AppName1 = 'Some Application'
$AppName1Regkey = 'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{AppName1-GUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = ({env:SystemDrive} + $errorpath)

# Delete leftover application Uninstall registry key
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 Uninstall registry key..."
Write-Host ""

if ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 Uninstall registry key exists."
    Write-Host ""
    Write-Host "Deleting $AppName1 Uninstall registry key..."
    Write-Host ""
    Write-Host "Command: Remove-Item -Path $AppName1Regkey -Recurse -Force"
    Write-Host ""

    Remove-Item -Path $AppName1Regkey -Recurse -Force

    Write-Host ""
    Write-Host "    $AppName1 Uninstall registry key deletion complete."
    Write-Host ""
}
else
{
    Write-Host "    $AppName1 Uninstall registry key does not exist."
    Write-Host ""
}

sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

Shortcuts - Copying and Creating

There are several ways to deal with application shortcuts:

- Install the application on a system, right-click the application executable, select **Create shortcut**, and then copy the resulting shortcut to the Software Vault to be copied into place during the installation process.
- Use WshShell to programmatically create the shortcut from a file on the target system.

Note: See the **File/Folder Deletion Methods** section for details on deleting a shortcut.

Copy an Existing Shortcut Using Copy-Item

```
# Copying an Existing Shortcut to the Public Desktop using 'Copy-Item' method

# Variables -- Modify
$AppName1 = 'Some Application'
$SourceExe1 = (${env:SystemDrive} + '\AppName1\AppName1.exe - Shortcut.lnk')
$DestinationPath1 = (${env:SystemDrive} + '\Users\Public\Desktop\AppName1.exe -
  Shortcut.lnk')
$AppName1Shortcut = 'AppName1.exe - Shortcut.lnk'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Copy existing shortcut to the Public Desktop
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1Shortcut shortcut on Public Desktop..."
Write-Host ""

If ((Test-Path -Path $DestinationPath1) -ne 'True')
{
    Write-Host "    $AppName1Shortcut shortcut on Public Desktop does not exist."
    Write-Host ""
    Write-Host "Copying $AppName1Shortcut shortcut to Public Desktop..."
    Write-Host ""
    Write-Host "Command: Copy-Item $SourceExe1 $DestinationPath1"

    Copy-Item $SourceExe1 $DestinationPath1 -Force

    Write-Host ""
    Write-Host "    Copying $AppName1Shortcut shortcut complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1Shortcut shortcut already exists."
    Write-Host ""
}

Sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

Creating a Shortcut Using WshShell

```
# Create a Shortcut on the Default User Desktop using 'wshShell' method

# Variables -- Modify
$AppName1 = 'Some Application'
$SourceExe1 = (${env:ProgramFiles(x86)} + '\AppName1\AppName1.exe')
$DestinationPath1 = (${env:SystemDrive} + '\Users\Default\Desktop\AppName1.lnk')
$AppName1Shortcut = 'AppName1.lnk'

# Variables -- Do NOT modify
$wshShell = New-Object -comObject WScript.Shell

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_InstallErrorLog.txt')
$errorFileLocation = (${env:SystemDrive} + $errorpath)

# Make a shortcut on the Default User Desktop
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1Shortcut shortcut on Default User Desktop..."
Write-Host ""

If ((Test-Path -Path $DestinationPath1) -ne 'True')
{
    Write-Host "    $AppName1Shortcut shortcut on Default User Desktop does not exist."
    Write-Host ""
    Write-Host "Creating $AppName1Shortcut shortcut on Default User Desktop..."
    Write-Host ""
    Write-Host "Command:  $Shortcut = $wshShell.CreateShortcut($DestinationPath1)"

    $Shortcut = $wshShell.CreateShortcut($DestinationPath1)

    Write-Host ""
    Write-Host "Command:  $Shortcut.TargetPath = $SourceExe2"

    $Shortcut.TargetPath = $SourceExe1

    Write-Host ""
    Write-Host "Command:  $Shortcut.Save()"

    $Shortcut.Save()

    Write-Host ""
    Write-Host "    Copying $AppName1Shortcut shortcut complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1Shortcut shortcut already exists."
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errorFileLocation
Write-Host "Log file location = $errorFileLocation"
Write-Host ""

timeout 5
```

Stopping Processes and Services

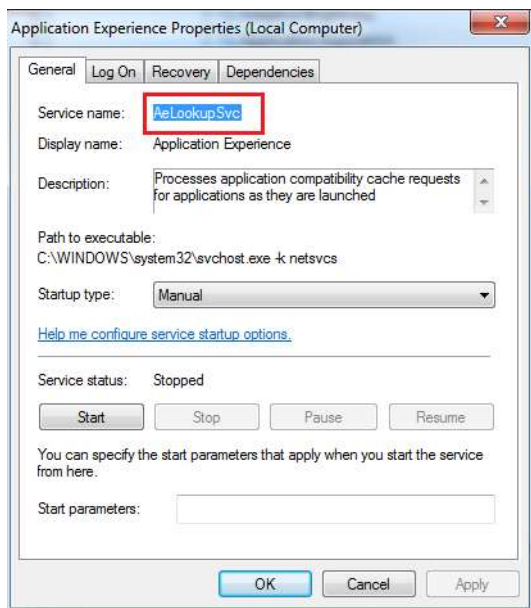
Uninstallers do not always stop an application's services and processes before uninstalling an application which always results in a failure.

If the uninstall process fails, open the **Services** management console and check for running services. Then, open **Task Manager** and check for running processes.

Services.msc

To find the name of a specific service do the following:

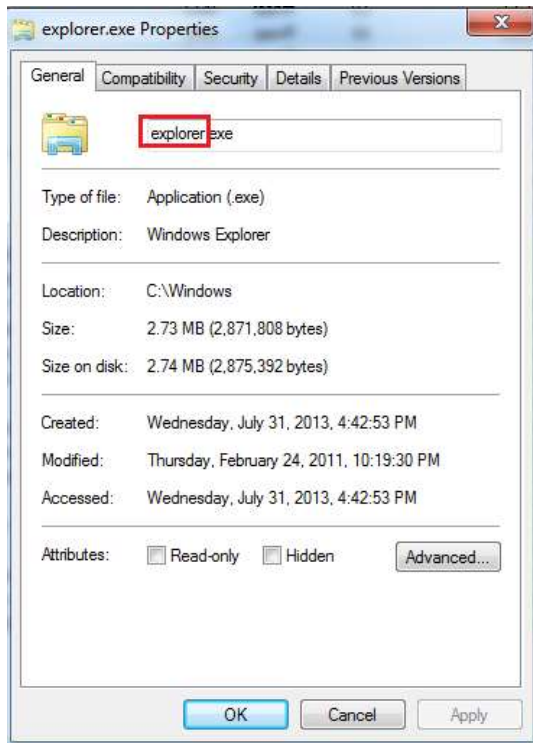
- Open the **Services** console.
- Double-click the service name in the listing to open the properties dialog.
- In the properties window, highlight the Service name and copy it into a PowerShell program variable.



Task Manager

To find the name of a specific process do the following:

- Open **Task Manager**.
- Select the **Process** tab.
- Find the running process in the **Name** listing.
- Highlight and right-click the process and the select **Properties**.
- On the **General** tab, copy the process name, minus the file extension, into a PowerShell program variable.



Uninstaller

```
# Stop-Process and Stop-Service before uninstalling AppName1

# Variables - Modify
$AppName1 = 'Some Application'
$AppName1DbSvc = "Some App Database"
$AppName1FileSvc = "Some App File Swap"
$AppName1UtilityMgr = "Some App Access Manager"
$AppName1Params = '/X {SomeApp-GUID} UNIQUE_NAME="foo" FULLNAME="Some Application"
ADDREMOVE=1'
$AppName1Regkey =
'HKLM:\SOFTWARE\wow6432Node\Microsoft\windows\CurrentVersion\Uninstall\{SomeApp-
GUID}'

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)
$MsiExec = (${env:SystemRoot} + '\System32\msiexec.exe')

# Uninstall AppName1 after first stopping the DB and File services, and the Utility
Manager process
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $AppName1 installation..."
Write-Host ""

# Check uninstall registry key or install path
If ((Test-Path -Path $AppName1Regkey) -eq 'True')
{
    Write-Host "    $AppName1 is installed."
    Write-Host ""
    Write-Host "Checking state of $AppName1DbSvc service..."
    Write-Host ""

    # Stop the AppName1 Database service
    If (((Get-Service $AppName1DbSvc).Status) -eq 'Running')
    {
        Write-Host "    $AppName1DbSvc service is running."
        Write-Host ""
        Write-Host "Stopping $AppName1DbSvc service..."
        Write-Host ""
        Write-Host "Command: Stop-Service $AppName1DbSvc -Force -Verbose"

        Stop-Service $AppName1DbSvc -Force -Verbose

        Write-Host ""
        Write-Host "    $AppName1DbSvc is stopped."
        Write-Host ""
    }
    Else
    {
        Write-Host "    $AppName1DbSvc service is not running."
        Write-Host ""
    }
}
Sleep 2

Write-Host "Checking state of $AppName1FileSvc service..."
Write-Host ""

# Stop the AppName1 File service
If (((Get-Service $AppName1FileSvc).Status) -eq 'Running')
{
    Write-Host "    $AppName1FileSvc service is running."
    Write-Host ""
    Write-Host "Stopping $AppName1FileSvc service..."
    Write-Host ""
    Write-Host "Command: Stop-Service $AppName1FileSvc -Force -Verbose"
```

```

        Stop-Service $AppName1FileSvc -Force -Verbose

        Write-Host ""
        Write-Host "    $AppName1FileSvc is stopped."
        Write-Host ""
    }
    Else
    {
        Write-Host "    $AppName1FileSvc service is not running."
        Write-Host ""
    }
    Sleep 2

    Write-Host "Checking state of $AppName1UtilityMgr process..."
    Write-Host ""

    # Stop the AppName1 Utility Manager process
    $ProcessActive = Get-Process $AppName1FileSvc -ErrorAction SilentlyContinue

    if($ProcessActive -ne $null)
    {
        Write-Host "    $AppName1UtilityMgr process is running."
        Write-Host ""
        Write-Host "Stopping $AppName1UtilityMgr process..."
        Write-Host ""
        Write-Host "Command: Stop-Process -Force -Name $AppName1UtilityMgr -Verbose"

        Stop-Process -Force -Name $AppName1UtilityMgr -Verbose

        Write-Host ""
        Write-Host "    $AppName1UtilityMgr process is stopped."
        Write-Host ""
    }
    Else
    {
        Write-Host "    $AppName1UtilityMgr process is not running."
        Write-Host ""
    }
    Sleep 2

    # Uninstall AppName1 using MSIEXEC with parameters
    Write-Host ""
    Write-Host "Uninstalling $AppName1..."
    Write-Host ""
    Write-Host "Command: $MsiExec $AppName1Params"
    Write-Host ""

    Start-Process -FilePath $MsiExec -ArgumentList $AppName1Params -ErrorVariable +err
    -Verb Open -Wait

    Write-Host ""
    Write-Host "    $AppName1 uninstall complete."
    Write-Host ""
}
Else
{
    Write-Host "    $AppName1 is not installed."
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5

```

32-Bit ODBC Connections

Some applications require a 32-bit ODBC database connection that uses an authenticated user's credentials to connect to a remote database. It is possible to create the database connection by using PowerShell to create the correct registry keys with values obtained from a screenshot of a valid user's existing database connection as with the MSOW package example below. A technician can then install the application package on a user's system without the user needing to provide credentials. When the user logs on to the system and authenticates, the database connection will already be in place to connect the user to the database. All the user must do is start the application as all the required pieces are in place.

MSOW_PRD:

Name	Type	Data
ab (Default)	REG_SZ	(value not set)
ab Database	REG_SZ	MSOW_PRD
ab Description	REG_SZ	MSOW Production
ab Driver	REG_SZ	C:\WINDOWS\system32\SQLSRV32.dll
ab LastUser	REG_SZ	User
ab Server	REG_SZ	msow-sql
ab Trusted_Connec...	REG_SZ	Yes

ODBC Data Sources:

Name	Type	Data
ab (Default)	REG_SZ	(value not set)
ab EasyLobby_TST	REG_SZ	SQL Server
ab EL100	REG_SZ	Microsoft Access Driver (*.mdb)
ab MSOW_PRD	REG_SZ	SQL Server

Installer

```
# Creating a 32-bit ODBC connection through registry key manipulation

# Variables -- Modify
$AppName1 = 'MSOW Application'
$ConnectionName = 'MSOW_PRD'
$ConnectionDescription = 'MSOW PRODUCTION'
$SqlServer = 'MSOW-SQL'
$SqlDatabase = 'MSOW_PRD'

# Variables -- Do NOT modify
$SqlDriver = '"C:\WINDOWS\System32\sqlsrv32.dll"'
$HKLMPath1 = "HKLM:SOFTWARE\wow6432Node\ODBC\ODBC.INI\" + $ConnectionName
$HKLMPath2 = "HKLM:SOFTWARE\wow6432Node\ODBC\ODBC.INI\ODBC Data Sources"

# Error file
$startLocation = Get-Location
$err=@()
$errorpath = ('\ ' + $AppName1 + '_InstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)

# Create a 32-bit ODBC connection
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $ConnectionName 32-bit ODBC connection..."
Write-Host ""

If ((Test-Path -Path $HKLMPath1) -ne 'True')
{
    Write-Host "    $ConnectionName 32-bit ODBC connection does not exist."
    Write-Host ""
    Write-Host "Creating $ConnectionName 32-bit ODBC connection..."
    Write-Host ""

    md $HKLMPath1 -ErrorAction silentlycontinue
    set-itemproperty -path $HKLMPath1 -name Driver -value $SqlDriver
    set-itemproperty -path $HKLMPath1 -name Description -value $ConnectionDescription
    set-itemproperty -path $HKLMPath1 -name Server -value $SqlServer
    set-itemproperty -path $HKLMPath1 -name LastUser -value ""
    set-itemproperty -path $HKLMPath1 -name Trusted_Connection -value "Yes"
    set-itemproperty -path $HKLMPath1 -name Database -value $SqlDatabase

    # Required to display the ODBC connection in the ODBC Administrator application.
    md $HKLMPath2 -ErrorAction silentlycontinue
    set-itemproperty -path $HKLMPath2 -name "$ConnectionName" -value 'SQL Server'

    Write-Host ""
    Write-Host "    $ConnectionName 32-bit ODBC connection creation complete."
    Write-Host ""
    Write-Host "To confirm 32-bit ODBC connection creation, run odbcad32.exe from the"
    Write-Host "    C:\windows\SysWOW64 folder"
    Write-Host ""
}
Else
{
    Write-Host "    $ConnectionName 32-bit ODBC connection already exists."
    Write-Host ""
}

sleep 5

# write error file
$err | Out-File $errFileLocation
Write-Host "Log file location = $errFileLocation"
Write-Host ""

timeout 5
```

Uninstaller

Deleting a 32-bit ODBC connection through registry key manipulation with user prompt

Variables -- Modify

```
$AppName1 = 'MSOW Application'
$ConnectionName = 'MSOW_PRD'
$ConnectionDescription = 'MSOW PRODUCTION'
$SqlServer = 'MSOW-SQL'
$SqlDatabase = 'MSOW_PRD'
```

Variables -- Do NOT modify

```
$SqlDriver = '"C:\WINDOWS\System32\sqlsrv32.dll"'
$HKLMPath1 = "HKLM:SOFTWARE\wow6432Node\ODBC\ODBC.INI\" + $ConnectionName
$HKLMPath2 = "HKLM:SOFTWARE\wow6432Node\ODBC\ODBC.INI\ODBC Data Sources"
```

Error file

```
$startLocation = Get-Location
$err=@()
$errorpath = ('\' + $AppName1 + '_UninstallErrorLog.txt')
$errFileLocation = (${env:SystemDrive} + $errorpath)
```

Delete the 32-bit ODBC connection

```
set-location $startLocation
Write-Host "*****"
Write-Host ""
Write-Host "Checking for $ConnectionName 32-bit ODBC connection..."
Write-Host ""
```

```
If ((Test-Path -Path $HKLMPath1) -eq 'True')
{
```

```
    Write-Host "    $ConnectionName 32-bit ODBC connection exists."
    Write-Host ""
    $title = "Remove $ConnectionName ODBC Connection"
    $message = "Do you want to remove the $ConnectionName 32-bit ODBC connection?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
        "Remove $ConnectionName connection"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
        $ConnectionName connection"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($yes, $no)
    $result = $host.ui.PromptForChoice($title, $message, $options, 0)
    Write-Host ""
```

```
    Switch ($result)
    {
        0 {"    You selected Yes."}
        1 {"    You selected No."}
    }
```

```
    If ($result -eq 0)
    {
```

```
        Write-Host ""
        Write-Host "Removing $ConnectionName 32-bit ODBC connection..."
        Write-Host ""
```

```
        Remove-Item -Path $HKLMPath1 -Force
```

Required to remove the ODBC connection in the ODBC Administrator app.

```
        Remove-ItemProperty -path $HKLMPath2 -name $SqlDatabase -Force
```

```
        Write-Host ""
        Write-Host "    Removal of $ConnectionName 32-bit ODBC connection complete."
        Write-Host ""
        Write-Host "To confirm 32-bit ODBC connection removal, run odbcad32.exe from
            the C:\windows\SysWOW64 folder"
        Write-Host ""
```

```
    }
    Else
    {
```

```
        Write-Host ""
```

```

        Write-Host "    Skipping $Shortcut 32-bit ODBC connection removal."
        Write-Host ""
    }
}
Else
{
    Write-Host "    $ConnectionName 32-bit ODBC connection does not exist."
    Write-Host ""
}

Sleep 5

# Write error file
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5

```

The SCCM Application Packaging how-to documentation continues in:

SCCM Application Packaging – Part 3 – Examples