2015

SCCM Application Packaging

A ZEN GUIDE TO SCCM APPLICATION PACKAGING WITH POWERSHELL

JEROLD "ZEN" SENFF

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

Contents

Potential Packaging Candidates

Every software application is a potential candidate for Microsoft System Center Configuration Manager (SCCM) distribution. Often, the only way to know if the application can be packaged is to actually install the application and see what happens. Applications can install, but not run on Microsoft Windows 7, or they may install and run, but not uninstall.

At the contract start, a local business had around 65 applications in their SCCM software library. Of the 65 existing applications, maybe 15 worked correctly in that the applications could uninstall successfully through SCCM. Most of the 65 applications did not have a SCCM uninstall method in place or had bad detection methods preventing SCCM from knowing the application even existed on a client system. Now, each of the 550-plus SCCM application packages successfully installs, detects the application's existence, and uninstalls the application along with all components.

This guide covers the application packaging process developed over a ten month period of packaging applications daily. The target audience is IT Professionals with little or no previous System Center Configuration Manager, application packaging, or PowerShell experience. Screenshots are included in the how-to package section to provide clarity. The PowerShell examples are written simply in a modular fashion so snippets may be copied from the document into a PowerShell ISE window to easily build a PowerShell install or uninstall script. All of the PowerShell `Start-Process` commands are executed with the `-Wait` switch since the Windows 7 operating system only allows one application to install at a time, though the application being installed may call other installers during the setup process. An effort was made to make the script execution window output look uncluttered so technicians could more easily track the progress of the application installation. The process presented in this guide worked successfully for packaging over 550 software applications.

Packaging Process

Every application is installed a minimum of three times during the packaging process:

- First time, manually install the application from a **Command Prompt (Admin)** window:
  - Execute the setup program in the command window with a `/?` (or `-?`)to see if there are any switches available supporting silent installation.
    - Note any help dialog that appears and take a screenshot for the OneNote document.
    - Open a browser and search using the application name followed by 'silent install' to see if anybody else has had any success silently installing the application.
  - Basic questions to ask:
    - Does the application install?
    - Does the application run?
    - Are shortcuts created on the **Desktop** or in the **Start Menu** for the user?
  - Documenting the application:
    - Create a OneNote document (or preferred document software) using the packaging template.
    - Open the **Programs and Features** control panel applet and sort by the **Installed On** column:
      - Does the application register?
      - How many other components were install with the application?
    - Open the registry editor (`regedt32`) and obtain the following information for the application and every component the application installs:
      - Uninstall registry key path
      - **UninstallString** value
      - **DisplayName** value
      - **DisplayVersion** value
      - **Publisher** value
  - Uninstall the application and any components:
    - If the UninstallString is an `.EXE` or the application installs extra components:
      - Create a PowerShell script to remove the application and components.
      - Execute the script to ensure it works correctly.
      - Check **Programs and Features** to ensure the program no longer displays.
      - Check to ensure the shortcuts created by the install get removed.
      - Check for leftover installation folders.
    - If the **UninstallString** is a `.MSI`, copy the string to the Administrator command window and execute:
      - Check **Programs and Features** to ensure the program no longer displays.
      - Check to ensure the shortcuts created by the install get removed.
      - Check for leftover installation folders.
- Second time, attempt to install the application silently:
  - Are shortcuts created on the **Desktop** or **Start Menu** for the user?
  - Do all components install correctly?
  - Does the application run?
  - Does the application uninstall silently?
  - If there is a PowerShell script for uninstalling the application, check the script again.
- Package, distribute, and deploy the application using the SCCM Console.
- Third time, test the SCCM application package through the **Software Manager** interface on the application packaging workstation:
  - Are shortcuts created on the **Desktop** or **Start Menu** for the user?

- Do all components install correctly?
- Does the application run?
- Does the application uninstall?
- Are the shortcuts removed?
- Are the installation folders removed?

Shared Components

Many applications require certain shared components to be install to for the application to execute correctly. Shared components include redistributables and runtime libraries. Ideally, it is best to package any shared redistributables and runtimes for deployment through SCCM and to use an install dependency in the application package to install the shared component. Often, an application setup will install a specific redistributable or library version, regardless of the existence of the newest version on the same system, which complicates managing shared components through SCCM.

The solution implemented at the local Seattle hospital has all the latest versions of shared redistributables and runtime libraries packaged for SCCM distribution. Older versions of the .NET Framework are also packaged. When an application setup installs a down-level version of a redistributable or runtime, the information is added to the *Application Dependency* tracking spreadsheet and the PowerShell uninstall script will prompt the technician on whether or not the redistributable or runtime should be uninstalled.

Blocked Applications

Applications can be blocked from packaging for a variety of reasons. The number one reason at the local Seattle hospital was a lack of setup files for applications.

Blocked applications - Examples:

- No setup files available: 165 apps (73%)
- Windows 7 incompatibility: 32 apps (14%)
- No license key available: 7 apps (3%)
- SCCM specific issue: 3 apps (1%)
- Requires CD-ROM: 3 apps (1%)
- Installer Issues: 4 apps (2%)
- Blocked by Group Policy: 6 apps (3%)
- Miscellaneous: 8 apps (3%)

Required Packaging Information

Information Required to Successfully Package an Application for SCCM Distribution

- The location of setup files for the application:
  - If obtaining from a web download, include a website link.
    - If a user account is required to download the application, an onsite client employee should obtain the setup files rather than the technician.
  - If obtaining from a network share, read/write permissions will be required to copy the setup files to the SCCM **Software Vault** share for packaging.
- Any information or special instructions regarding the following:
  - License keys
  - User name
  - Company name
  - File/folder copying
  - Shortcuts
  - Network share connections
  - Database connections
  - Ports
  - Authentication types
  - File/folder ACL changes
  - Security group assignments
  - Site servers
  - Any other items requiring additional PowerShell development time to install and/or uninstall
- Access to network shares and/or databases will be needed not only to successfully install the program to create the SCCM package, but for any technicians that will be installing the package on an end-user's system.
  - A screenshot of the relevant registry key containing the connection information can also be used to create the key and settings directly with PowerShell.

Application Package Licensing

- If the application is a single license package, the license key will not be included with the SCCM package, but will be required to test the SCCM package to ensure the package installs correctly.
- If the application has a site, group, or enterprise license, the license will be included with the SCCM package unless otherwise specified. This normally requires using the software publisher's tool or methodology to embed the license key in the application's setup files.
- Installs requiring license keys to complete setup and/or user names and passwords to access any resources will be packaged with "**- Manual**" in the package name, and a technician will be required to manually enter the information during the setup process.

General Rules

- If the application cannot be installed successfully on the desktop test system, it cannot be packaged for SCCM distribution.
- Not all applications can be automated to install without user interaction.
- If an application installs multiple software components, all components will be uninstalled using a PowerShell script requiring additional development time.
    - The technician uninstalling the application will be prompted on whether or not to uninstall shared software components such as Microsoft Visual C++ Redistributables and runtime libraries.
- Each environmental configuration change made during an installation using PowerShell will need to be reversed during an uninstall using PowerShell, which requires additional development time.

Documentation Produced

Application Packaging tracks and documents the following items:

- OneNote document (or preferred document software) created in the Packaging section for every packaged application:
    - See the " Packaging Template" document below for an example.
- OneNote document (or preferred document software) created in the Reference section for every PowerShell script:
    - Install scripts
    - Uninstall scripts
    - Detection methods
- SCCM packaged applications inventory spreadsheet with applications sorted by name and categories (e.g., redistributables, multimedia, research, productivity, etc.).
- Application packaging software dependencies between various applications to identify which items should prompt the technician during an uninstall. Examples:
    - Adobe Acrobat
    - Adobe Flash Player
    - Apple Bonjour
    - Microsoft .NET Frameworks
    - Microsoft Product Runtime releases
    - Microsoft Visual C++ Redistributables
- Blocked applications - Examples:
    - No setup files available
    - Incompatible with Windows 7
    - Installs but does not start
    - Installs but does not uninstall
    - Needs newer version
    - No license key available to complete setup
    - Requires CD to run
- Applications on hold that should not be packaged until authorized:
    - Local Seattle hospital:
        - SQL Server releases blocked by Group Policy
        - VPN Client

Packaging Template

Every application packaged for SCCM distribution will have a corresponding OneNote document (or preferred document software) in the Packaging section containing the details required to successfully package each application in case the existing package is corrupted or deleted and needs to be repackaged quickly.

Template

Description
Brief product description.

Package
Free software license. Single license package. Group/Enterprise/Site license. Any specific instructions for installing the application including estimated time for large installs. Copy this paragraph to all the Notes/Comments sections in SCCM during package creation.

Source file location:

Copy source files to:

Import xxx into SCCM Applications as a xxx.

Install:

Uninstall:

Detection Method:

User Experience:
Behavior:
Logon:
Visibility:
Enforce: No specific action

Dependencies (list redistributables, runtime libraries, etc,. available through SCCM):

Notes
GUID:

Registry key:

UninstallString:

DisplayName:
DisplayVersion:
Publisher:

Note: Microsoft OneNote was already in use at this location for documentation. Any document software may be used. The key is being consistent in creating a "recipe" for every application packaged, including any PowerShell scripts for that package.

Status Email

A status email is generated to the SCCM Application Packaging team every day, or less often when packaging is slow. The email contains the current total packages in SCCM along with a breakdown of new applications packaged, existing packages tested, and application packages pending. Links to the various spreadsheets that the SCCM Application Packager maintains are also included along with their current numbers. There are currently three spreadsheets and a OneNote that are updated daily as part of the application packaging process.

Status Email Sample

To: SCCM Project Manager
CC: SCCM Team Members, Other Stakeholders
Subject: App Packaging Status

Total packages = 585

Created documentation in OneNote Reference section for updating the *ComplianceTest.ps1* script to create new Surface Pro 3 driver packages along with driver name/version troubleshooting steps.

New apps packaged:
- Adobe Flash Player 16 ActiveX
- Adobe Flash Player 16 Plugin
- ProcartaPlex Analyst - Manual

Existing packaged apps tested:
- Adobe Flash Player 15 ActiveX
- Adobe Flash Player 15 Plugin

Pending apps:
- None

The ***SCCM Packaged Apps - Categories*** spreadsheet has been updated to include all applications currently in SCCM:
- Spreadsheet is located at:
    - \\server\path\SccmPackaging\SCCM Packaged Apps - Categories.xlsx

Blocked apps: Total = 238
- Blocked apps have been broken out into a separate spreadsheet located at:
    - \\server\path\SccmPackaging\BlockedApps.xlsx, ***Blocked Applications*** sheet

Apps on hold: Total = 8
- Apps on hold have been broken out into a separate spreadsheet located at:
    - \\server\path\SccmPackaging\BlockedApps.xlsx, ***Apps on Hold*** sheet

Application Dependencies:
- Application dependencies are listed in the following spreadsheet located at:
    - \\server\path\SccmPackaging\AppPckgDependencyMap.xlsx

How to Package Applications

To package applications for SCCM distribution, the following client site domain user accounts are required:

- Domain user account (or equivalent) to log on for testing published application packages:
  - Add to onsite physical system's **Local Administrators** security group
- Domain engineer user account (or equivalent):
  - Read/write permissions for **Software Vault** share
  - Read/write permissions for all shares with application setup files
  - SCCM Security Roles:
    - Application Administrator
    - Desktop Technician
  - SCCM Role-Based Collection to test deployments:
    - IS Desktop Test

1. How to Obtain the Package Details

Software required:

- 7-Zip (http://www.7-zip.org/download.html)
- Orca (https://msdn.microsoft.com/en-us/library/windows/desktop/aa370557(v=vs.85).aspx)
- regedt32.exe (Located in the %SystemRoot%/System32 folder)
- Microsoft OneNote (or preferred document software)

In the *Application Packaging Notes* document, create a new page under *SCCM Packaging* to document the application being packaged.

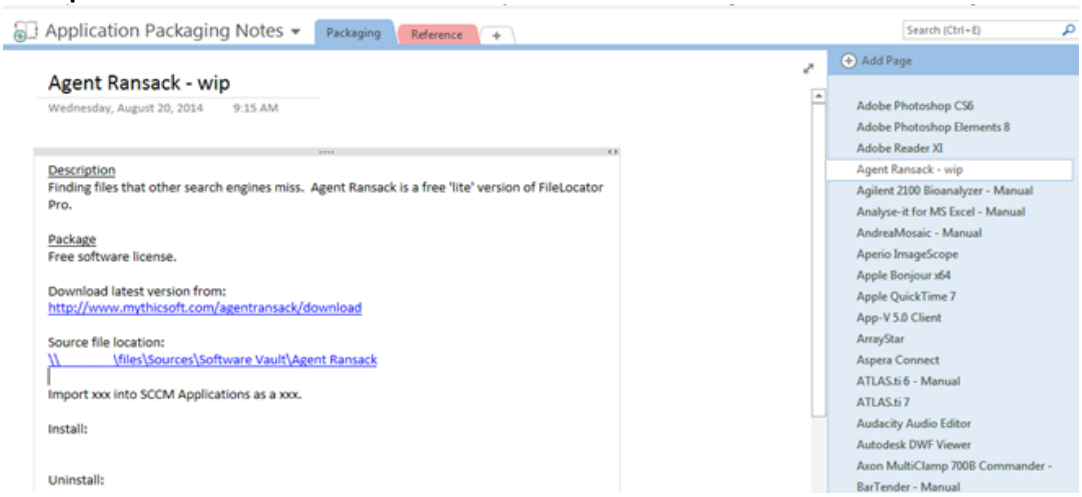
- Naming convention:
  - Applications not requiring user intervention: **AppName**
  - Application requiring user intervention: **AppName - Manual**
  - Applications being packaged need a **- wip** after the name to indicate a "work in progress"
- Copy the contents of the **Packaging Template** template into the newly created page (see *Packaging Template* section above).
- Be sure to fill out all the information while progressing through the packaging process.

Locate the application install folder with the setup file or the webpage containing a download link.

In the application packaging document:

- Find a brief description of the application for the **Description** section at the top of the page.
- Determine the type of licensing for the application:
  - Free software license - No purchase required.
  - Single software license - Every install will need a unique key to finish the setup wizard or to activate the product after installation.
  - Enterprise/Group/Site license - The license key is included in the installer and the user does not need to provide a license.
- Document the source of the setup file.
  - Internal share: Create a link containing the path to the internal share.
  - External download: Create a link to the download webpage.

Example:



- Find the install folder with the setup file(s), or the download webpage, and copy/download the setup file(s) to a new folder in the SCCM **Software Vault** share.
- Open **Explorer** and browse to the install folder containing the setup file.
- Right-click the setup file and select **7-Zip, Open archive** to determine if the setup file is a self-extracting zip with an MSI or a `setup.exe` file that can be extracted to the install folder. Extract the files if possible to the install folder.
- If the install file is not a self-extracting zip file, open a **Command Prompt (Admin)** window.
- Using the copied folder path from above, type **pushd** and paste in the folder path to switch to the setup folder on the Software Vault share and press **Enter**.

  ```
  pushd  \\server\share\Software Vault\AppName
  ```

- Type in the name of the install file followed by a forward-slash and question mark to determine if the setup file has any setup options for a quiet install. `-?` can also be tried.

  ```
  setup.exe /?
  ```

- If there are no setup options available, run the setup executable and stop at the setup wizard startup screen.
- Open Explorer and browse to the user **%temp%** folder and look for the setup files. For example:

  C:\Users*UserName*\AppData\Local\Temp\Mythicsoft\AgentRansack_7.0.822.1

- Copy all install folders in the **%temp%** folder to the install folder on the Software Vault share.
- Finish the setup wizard.

Start the application to make sure it installed correctly.

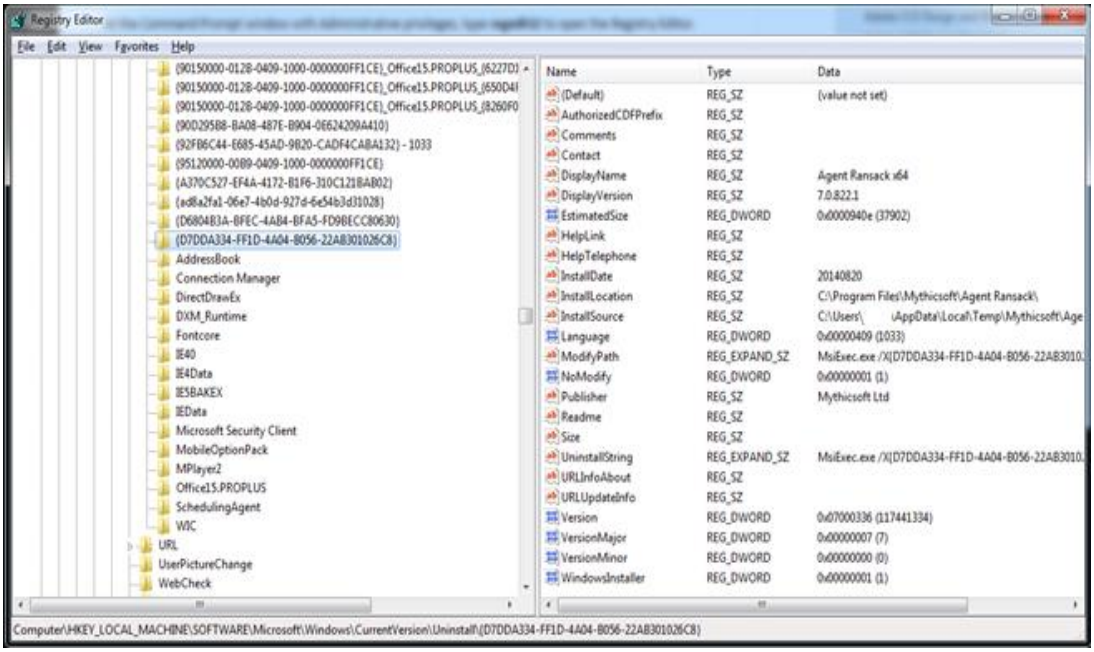
- A shortcut in the **Start Menu** or on the **Desktop** should be present and the application should open.
- Open the **Programs and Features** control panel applet and sort the **Installed On** column so the most recently installed programs are at the top.
- Make note of any newly installed applications like redistributables and runtime libraries.
- In the **Command Prompt(Admin)** window, type **regedt32** to open the Registry Editor.
- Find the Uninstall registry key for the application.
  - 64-bit applications will be found at:

    ```
    HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
    ```

  - 32-bit applications will be found at:

    ```
    HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall
    ```

Agent Ransack Example:



In the OneNote document:

- Scroll to the bottom of the OneNote application package page to the **Notes** section.
- Copy the registry key path and paste under **Registry key**.
- Copy the application GUID and paste after **GUID:** at the top of the **Notes** section and note whether the GUID is 32-bit or 64-bit.
- Copy the data from the **UninstallString** parameter value and paste under **UninstallString:**
- Copy the data from the **DisplayName** parameter and paste after **DisplayName:**
- Copy the data from the **DisplayVersion** parameter and paste after **DisplayVersion:**
- Copy the data from the **Publisher** parameter and paste after **Publisher:**

Example:

Notes
GUID: {D7DDA334-FF1D-4A04-B056-22AB301026C8} (64-bit)

Registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{D7DDA334-FF1D-4A04-B056-22AB301026C8}

UninstallString:
MsiExec.exe /X{D7DDA334-FF1D-4A04-B056-22AB301026C8}

DisplayName: Agent Ransack x64
DisplayVersion: 7.0.822.1
Publisher: Mythicsoft Ltd

If the **UninstallString** is an MsiExec call like above, you now have all the information needed to package the application.

The completed OneNote document for the Agent Ransack application package:

Description
Finding files that other search engines miss. Agent Ransack is a free 'lite' version of FileLocator Pro.

Package
Free software license.

Download latest version from:
http://www.mythicsoft.com/agentransack/download

Source file location:
\\ \files\Sources\Software Vault\Agent Ransack

Import *AgentRansack_822.exe* into SCCM Applications as a Script Installer.

Install:
"AgentRansack_822.exe"

Uninstall:
msiexec /x {D7DDA334-FF1D-4A04-B056-22AB301026C8} /qn /norestart

Detection Method:
MSI Product Code: {D7DDA334-FF1D-4A04-B056-22AB301026C8}

User Experience:
Behavior: Install for user
Logon: Only when a user is logged on
Visibility: Normal
Enforce: No specific action

Dependencies:
Office 2010 Filter Packs (64-bit)

Notes
GUID: {D7DDA334-FF1D-4A04-B056-22AB301026C8} (64-bit)

Registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{D7DDA334-FF1D-4A04-B056-22AB301026C8}

UninstallString:
MsiExec.exe /X{D7DDA334-FF1D-4A04-B056-22AB301026C8}

DisplayName: Agent Ransack x64
DisplayVersion: 7.0.822.1
Publisher: Mythicsoft Ltd

If the **UninstallString** calls an uninstall program, which may or may not take parameters, a PowerShell script may be required to uninstall the application. See the **PowerShell Scripting** section of this

document for more information about PowerShell script examples, snippets, detection methods, and install/uninstall methods.

Example:

UninstallString:
C:\Program Files (x86)\InstallShield Installation Information\{F60B80F1-7F44-4491-AD8D-7100A3F66A44}\setup.exe -runfromtemp -l0x0409

Uninstall the application by pasting the **UninstallString** into the **Command Prompt (Admin)** window and press **Enter** to uninstall the application. Check the following after the application uninstalls:

- Open the **Programs and Features** control panel applet to ensure that the application is no longer listed.
- Check the registry to ensure that the application **Uninstall** key was removed during the uninstall process.
- Open **Explorer** and browse to the application install folder to see if any leftover files or folders remain.
- Open the **Start Menu** and ensure that the application shortcuts were removed. Any **Desktop** shortcuts should also be removed.

Move on to the **2. How to Package MSI Installers** section or the **3. How to Package Script Installers** section.

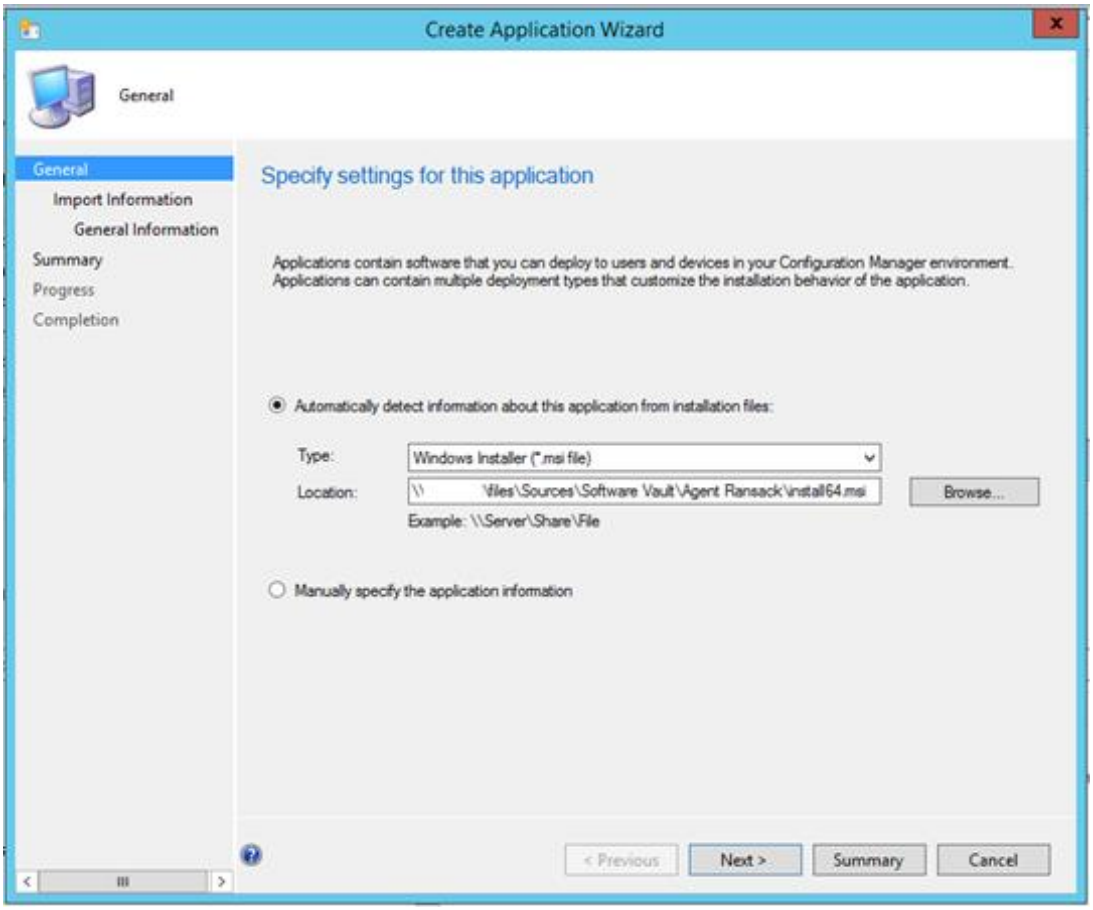
2. How to Package MSI Installers

To create the MSI Installer:
Open a Remote Desktop session to the server running SCCM and logon using a user account with appropriate permissions.

Start the **System Center 2012 Configuration Manager**.

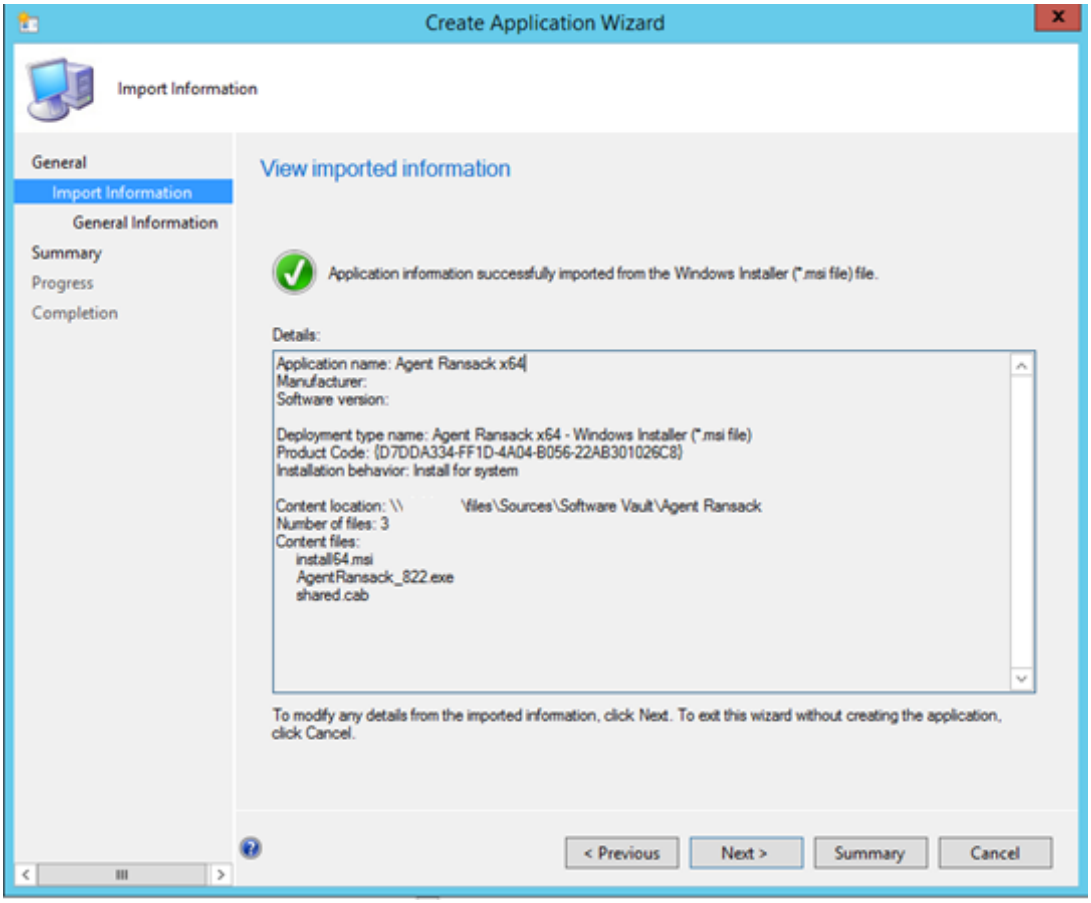
In the lower left corner, select **Software Library**, expand **Application Management**, expand **Applications**, right-click **Client Apps**, and select **Create Application**.

Paste the path to the install folder on the Software Vault share into the **Location:** field on the *Specify settings for this application* screen, and click the **Browse:** button. In the **Open** file wizard, select the .MSI file and click **Open**.



Click **Next** in the wizard, and if a message popup opens indicating that the publisher cannot be verified, click **Yes**.

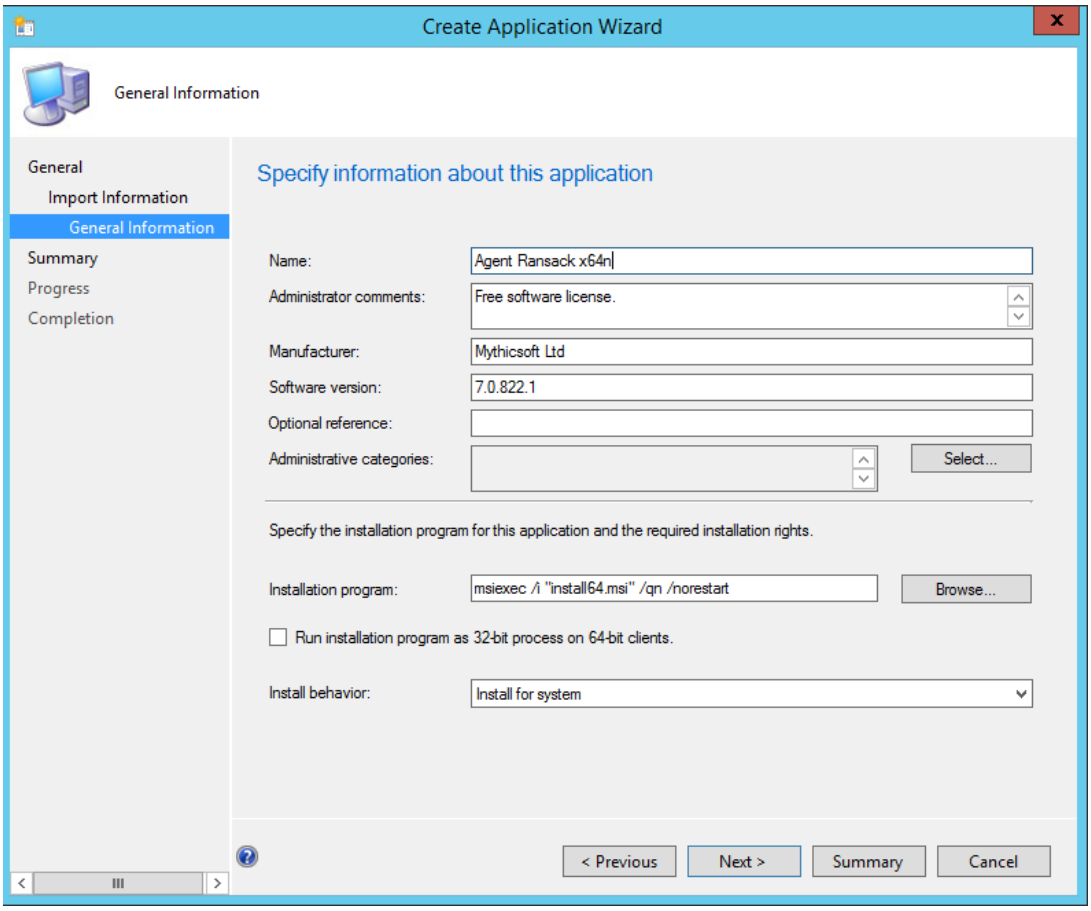
Click **Next** on the *View imported information* screen.



On the *Specify information about this application* screen, do the following with information from the application package document:

- Specify the application licensing type in the **Administrator comments:** field and include any other relevant details.
- Copy the **Publisher** value from Notes section into the **Manufacturer:** field.
- Copy the **DisplayVersion** value from the Notes section into the **Software version:** field.
- In the **Installation program:** field, ensure that the msiexec call has a **/qn /norestart** at the end:
  - Example: `msiexec /i "install64.msi" /qn /norestart`
- Ensure that the **Install behavior:** dropdown is set to **Install for system**.

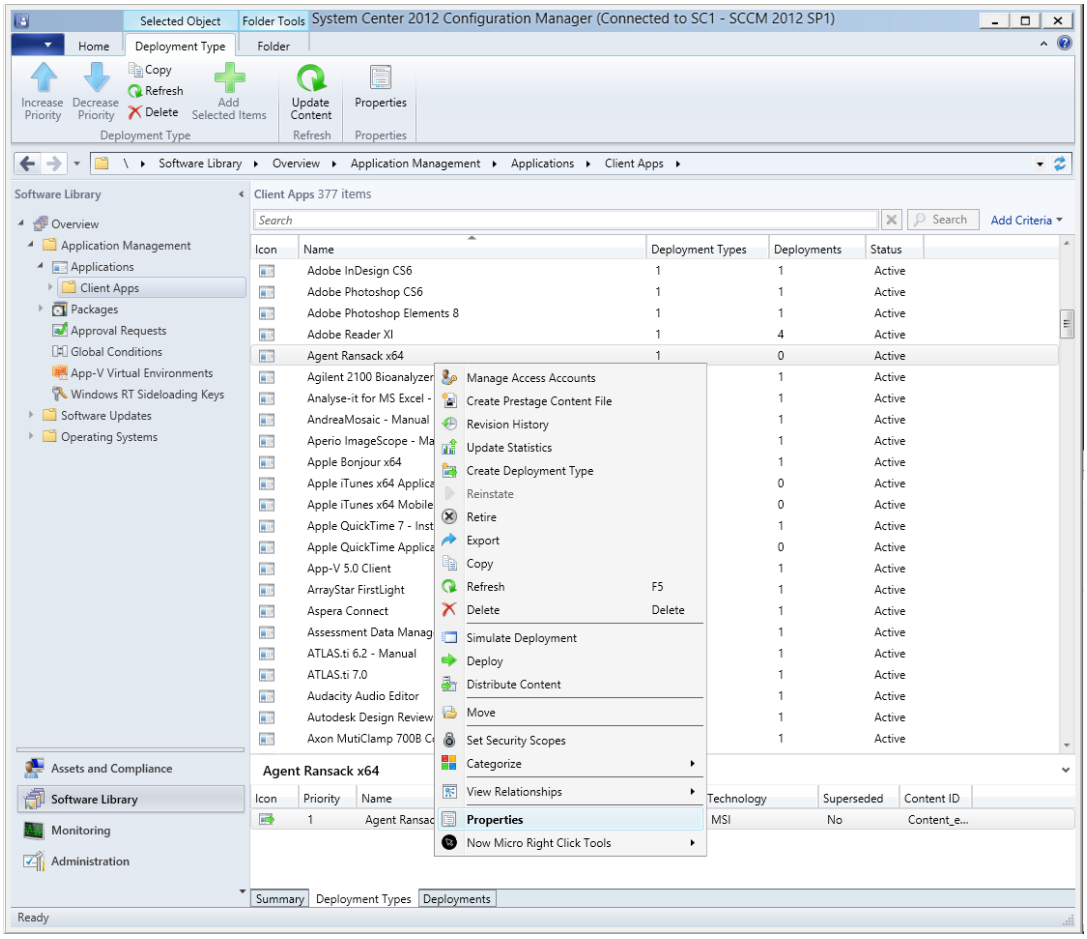
Click **Next** in the wizard.



Click **Next** at the *Confirm the settings for this application* screen.

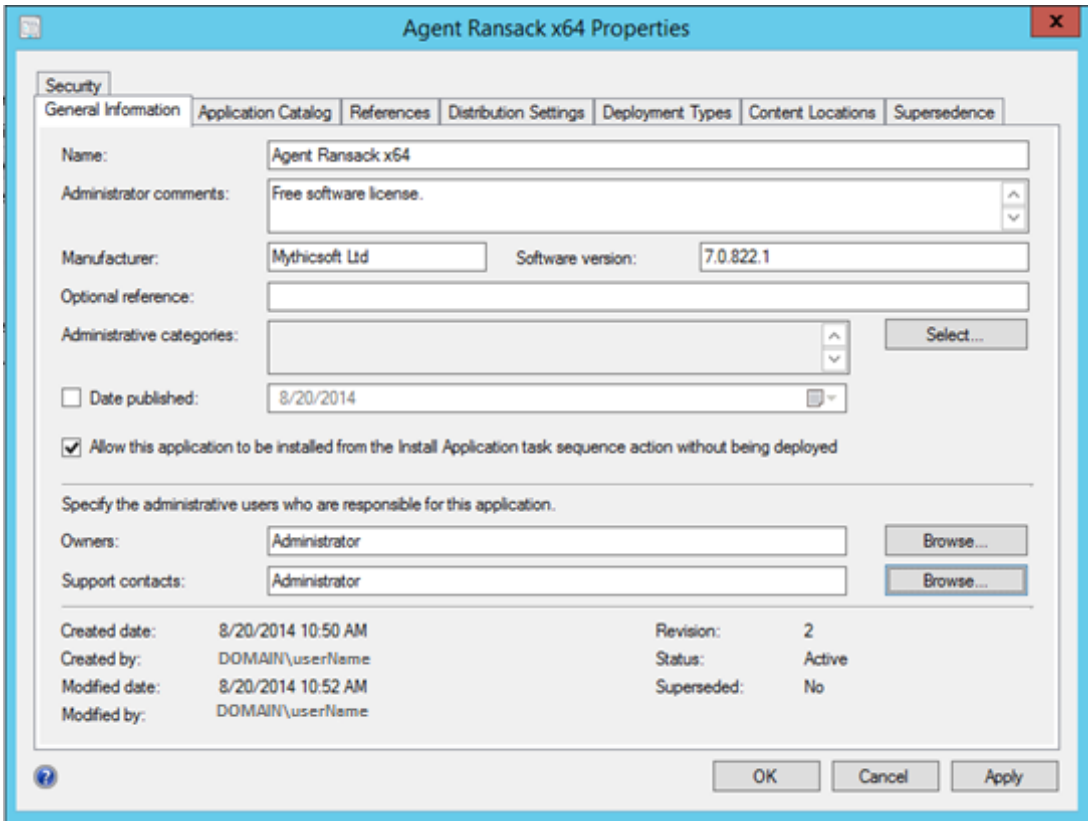
Click **Close** to exit the wizard.

In the SCCM **Client Apps** list, find the newly created package, right-click the name and select **Properties**.



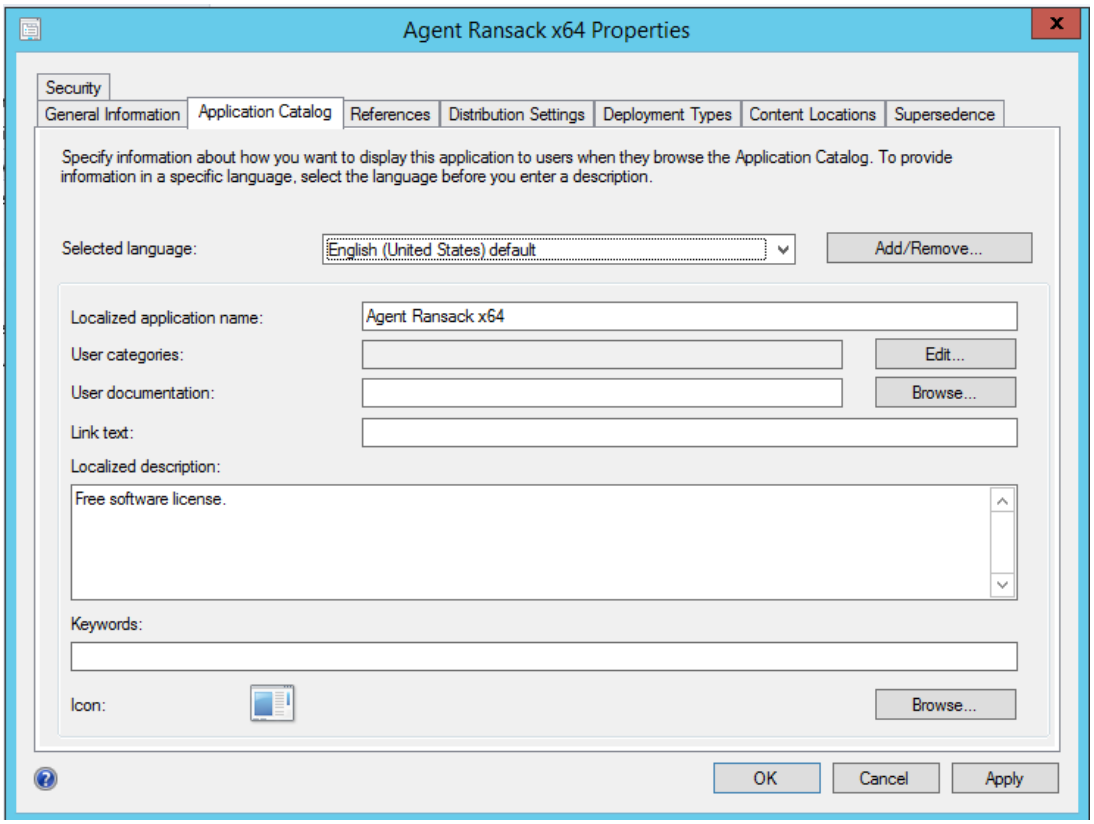
On the **General Information** tab:

- Enable the *Allow this application to be installed from the Install Application task sequence action without being deployed* checkbox.
- Change the **Owners:** field to **Administrator**
- Change the **Support contacts:** field to **Administrator**



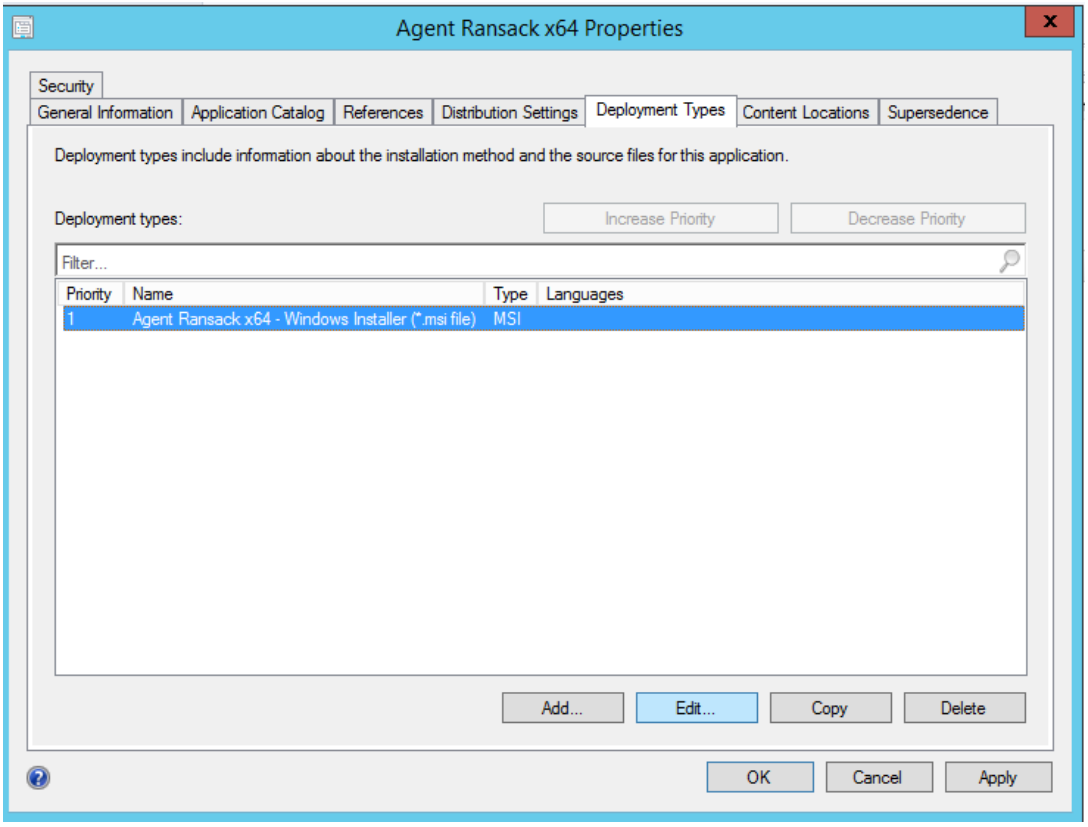
On the **Application Catalog** tab:

- Ensure that the **Localized description:** field indicates the application licensing type and any important installation notes.



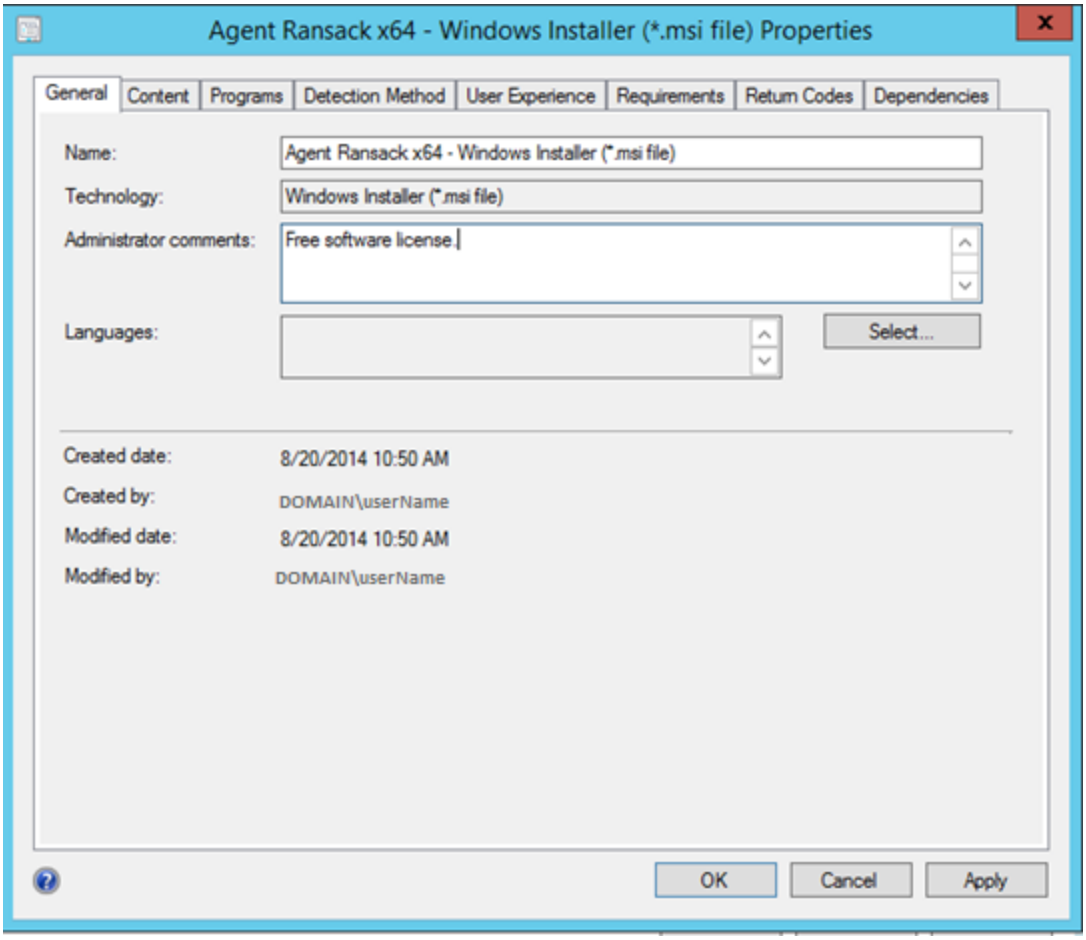
On the **Deployment Types** tab:

- Select the name of the MSI deployment type and click the **Edit...** button.

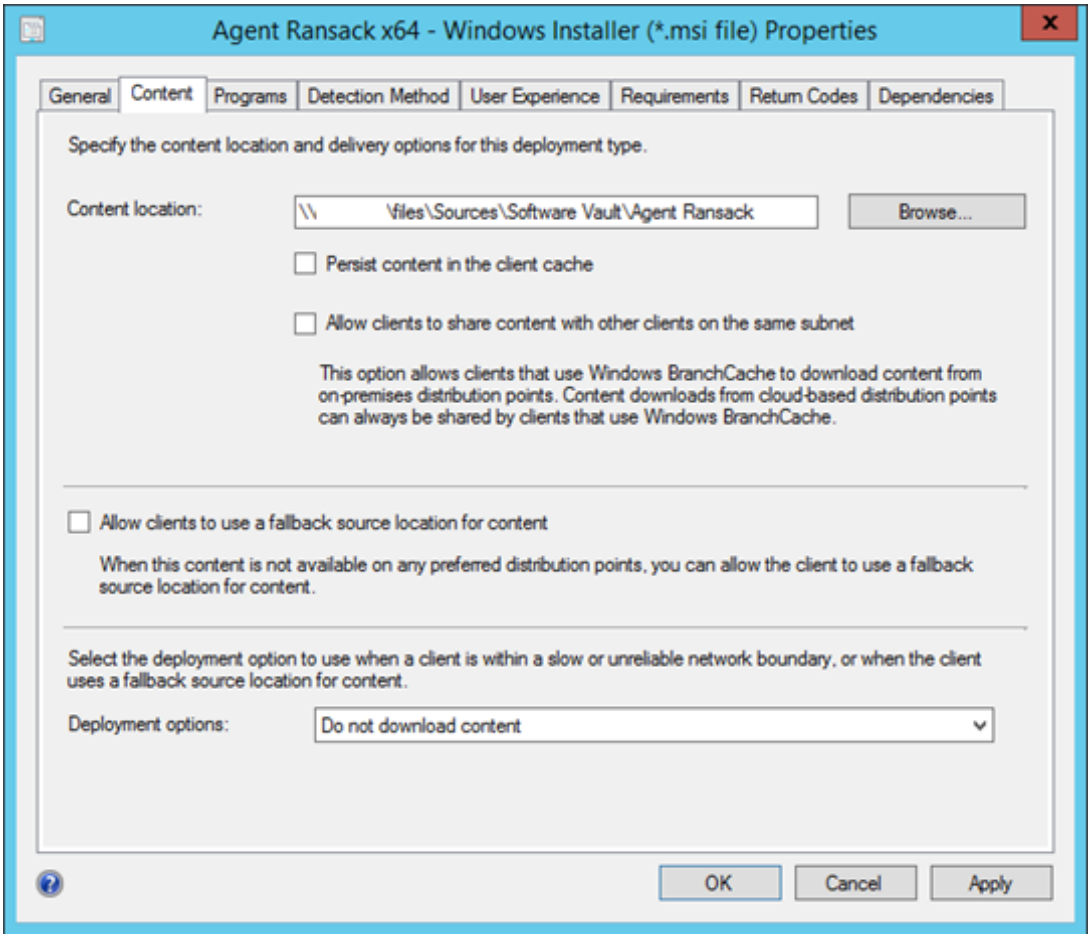


On *Windows installer (*.msi file) Properties* page, do the following:

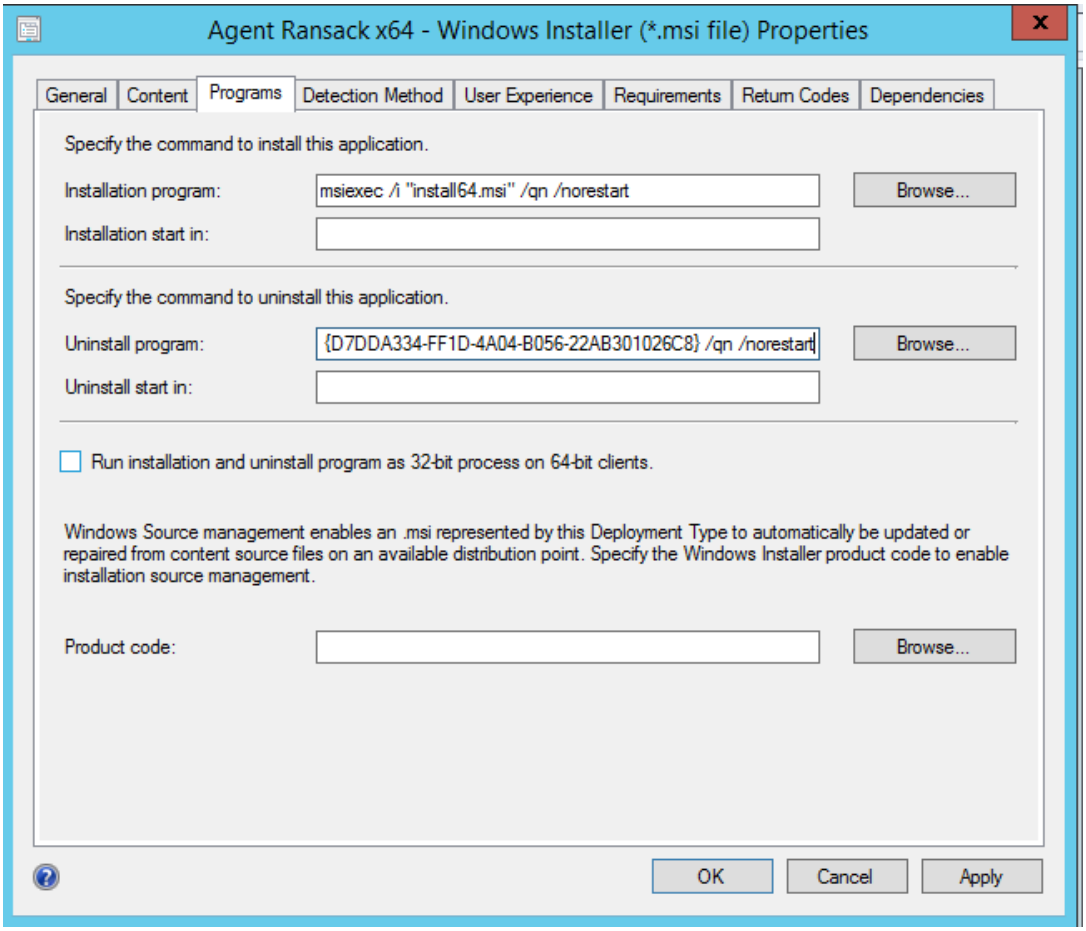
- On the **General** tab, add the application licensing type and any important installation notes to the **Administrator comments:** field.



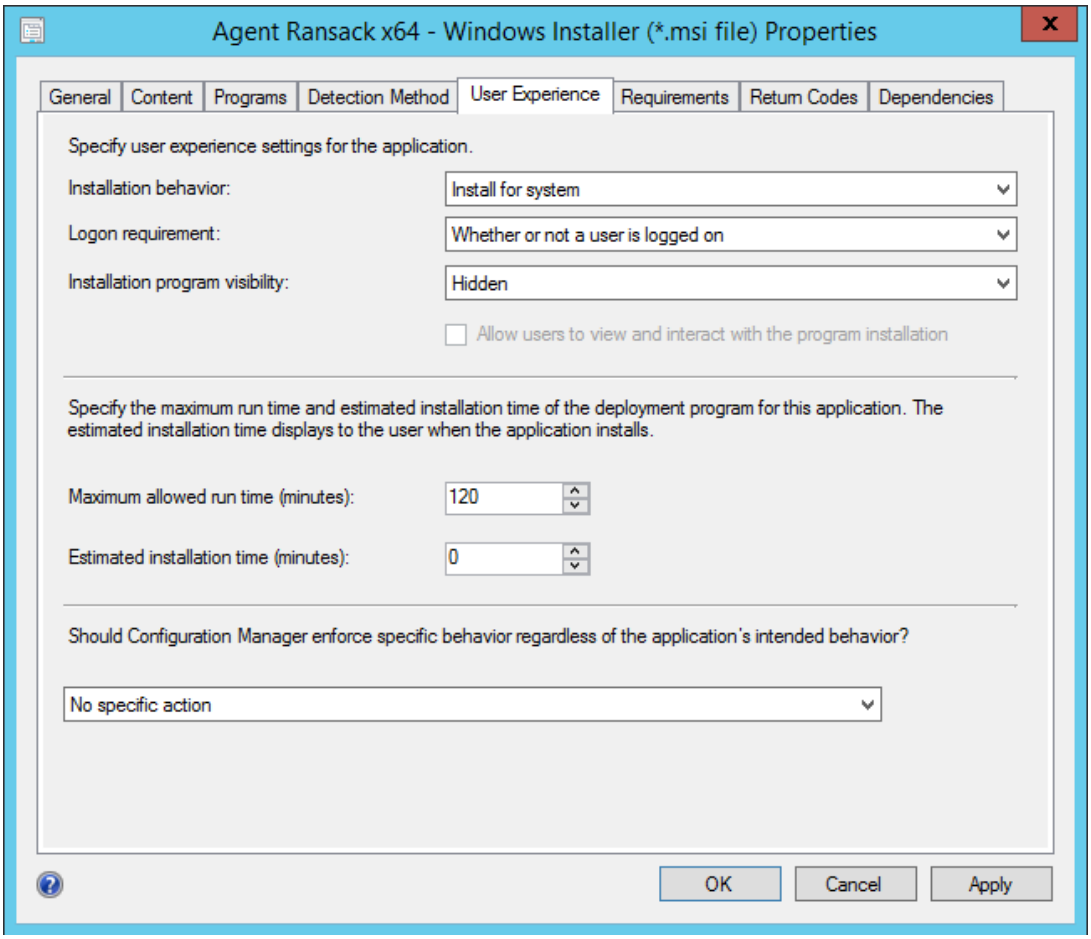
- On the **Content** page, uncheck the *Allow clients to share content with other clients on the same subnet* checkbox.



- On the **Programs** tab, add **/qn /norestart** to the end of the string in the **Uninstall program:** field.



- On the **User Experience** tab, do the following:
  - **Installation behavior:** should be set to **Install for system**
  - **Logon requirement:** should be set to **Whether or not a user is logged on**
  - **Installation program visibility:** should be set to **Hidden**
  - The application behavior dropdown should be set to **No specific action**



Click **OK** to close the *Windows Installer (*.msi file) Properties* page.

Click **OK** to close the application properties page.

Find the application package in the SCCM **Client Apps** list and move on to the **4. How to Distribute Content** section.

Note: If an application takes a long time to install, set **Visibility** to **Normal** and use the `/qb` switch with the `msiexec` command in the **Install program** field from the previous step to make the progress bar visible so technicians can see that the installation is running without having to open the Task Manager to search for the process.

3. How to Package Script Installers

Script installer packages are used for installers that use an executable file (`.EXE`) or require a PowerShell script. All PowerShell script calls need to be in the following format to execute properly on the client system.

```
Powershell.exe –executionpolicy Bypass –file "FileName.ps1"
```

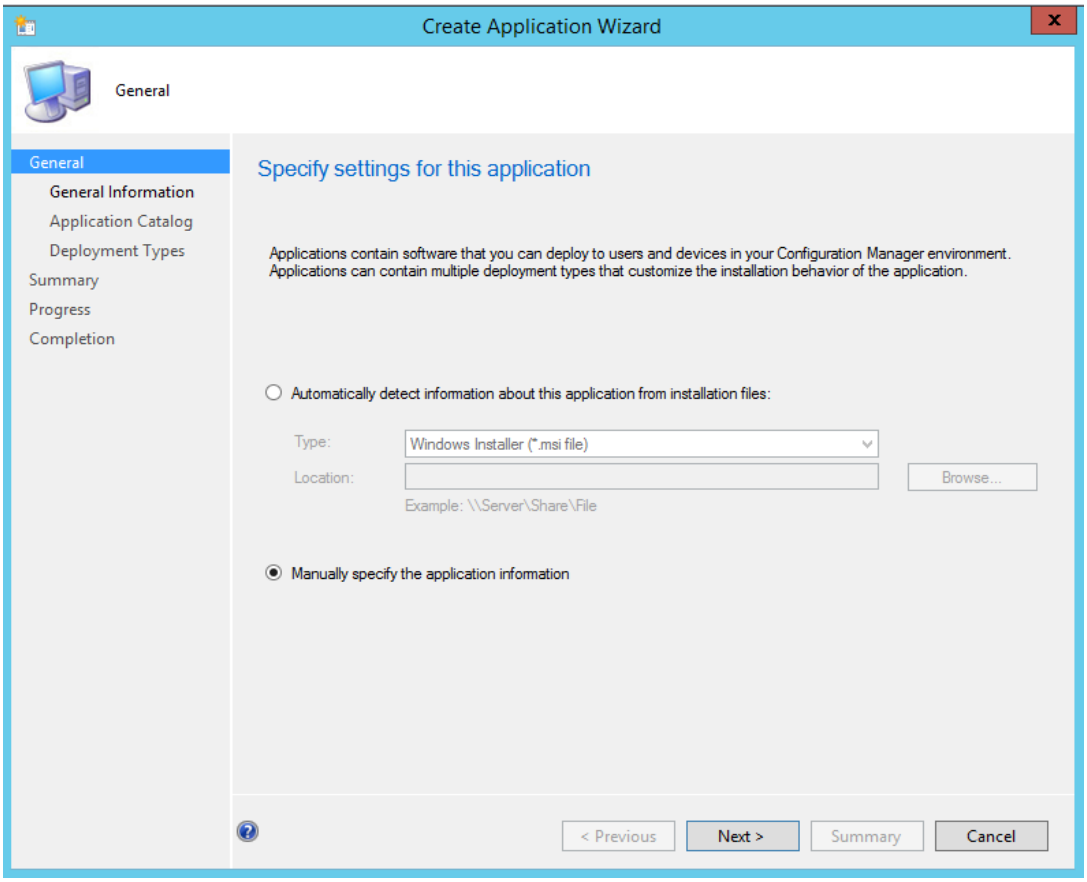
To create the Script Installer:

Open a Remote Desktop session to the server running SCCM and log in using a Domain Admin user account.

Start the **System Center 2012 Configuration Manager**.

In the lower left corner, select **Software Library**, expand **Application Management**, expand **Applications**, right-click **Client Apps** and select **Create Application**.

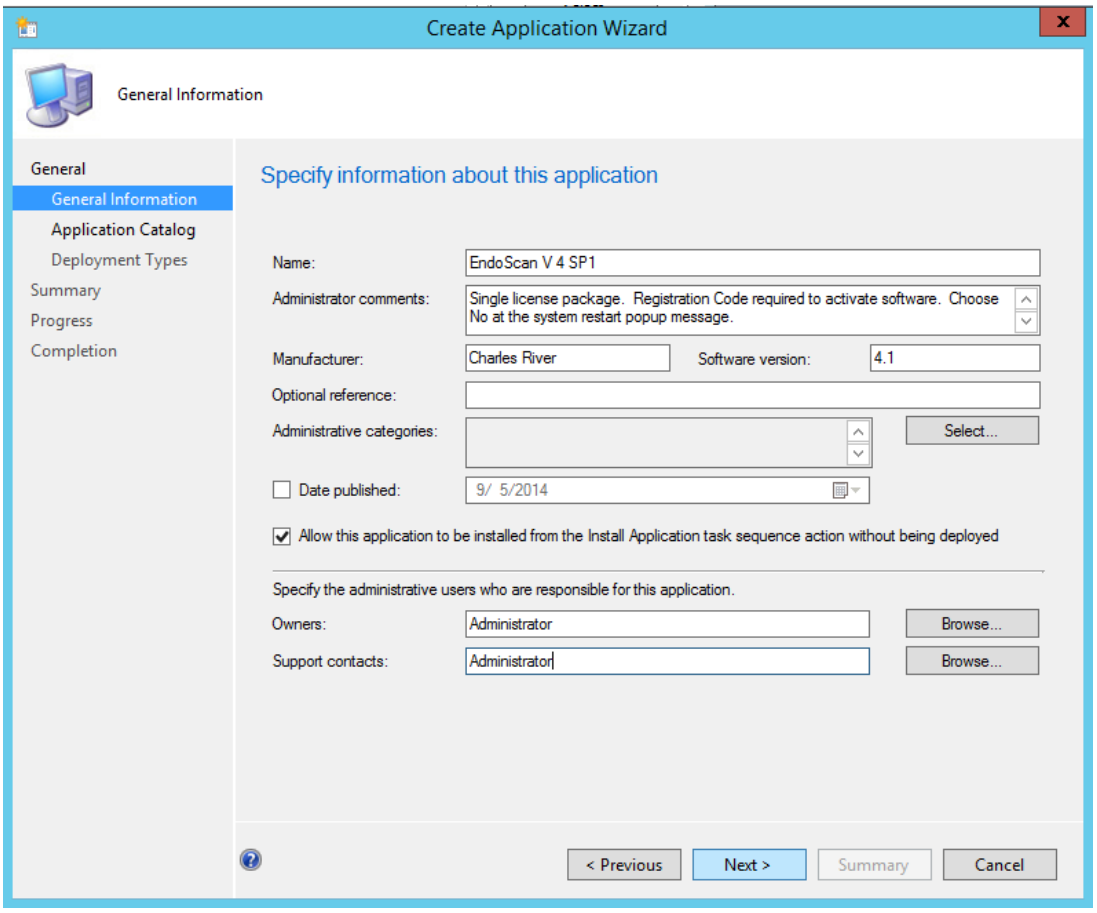
Select the **Manually specify the application information** button on the *Specify settings for this application* page. Click **Next** in the wizard.



On the *Specify information about this application* screen, do the following with information from the application package document:

- Specify the application's name in the **Name:** field. If the package requires user interaction to install, add **- Manual** after the name.
- Specify the application licensing type and any important installation notes in the **Administrator comments:** field.
- Copy the **Publisher** value from Notes section into the **Manufacturer:** field.
- Copy the **DisplayVersion** value from the Notes section into the **Software version:** field.
- Enable the **Allow this application to be installed from the Install Application task sequence without being deployed** checkbox.
- Change the name in the **Owners:** field to **Administrator**
- Change the name in the **Support contacts:** field to **Administrator**

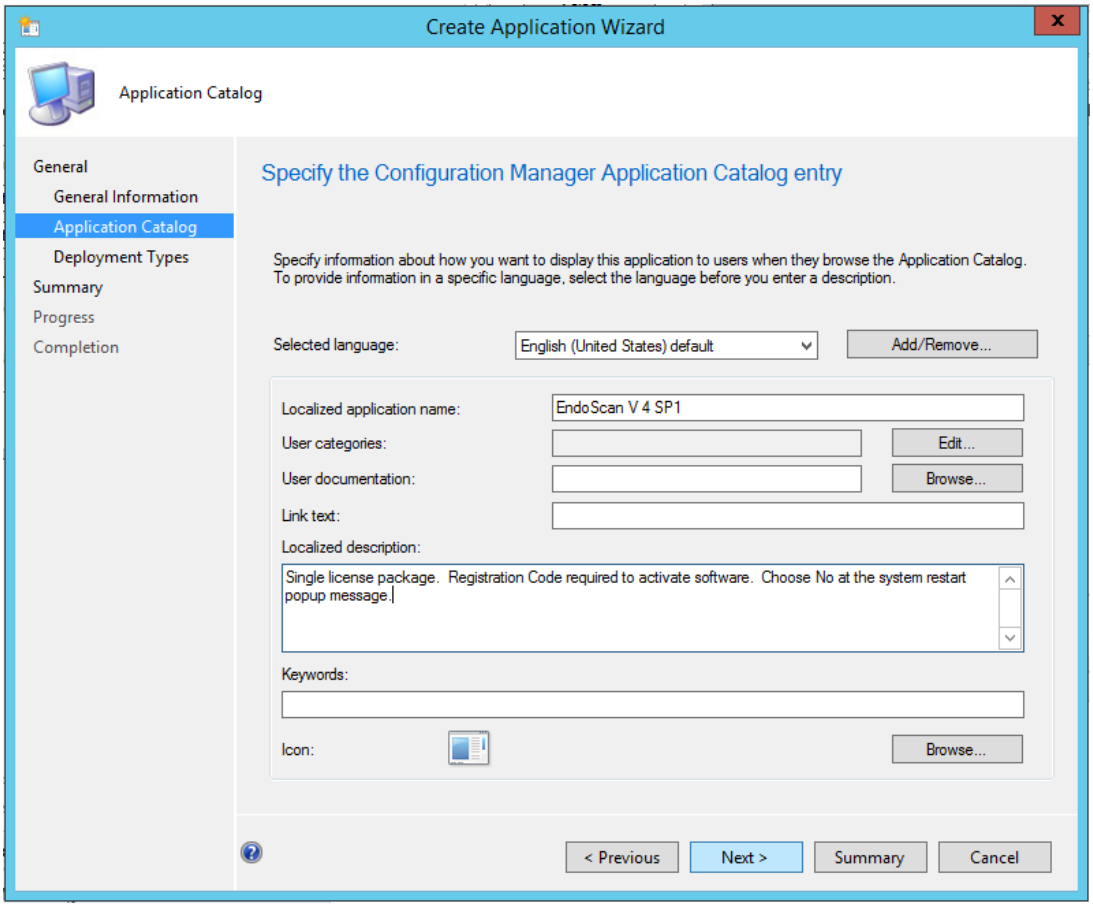
Click **Next** in the wizard.



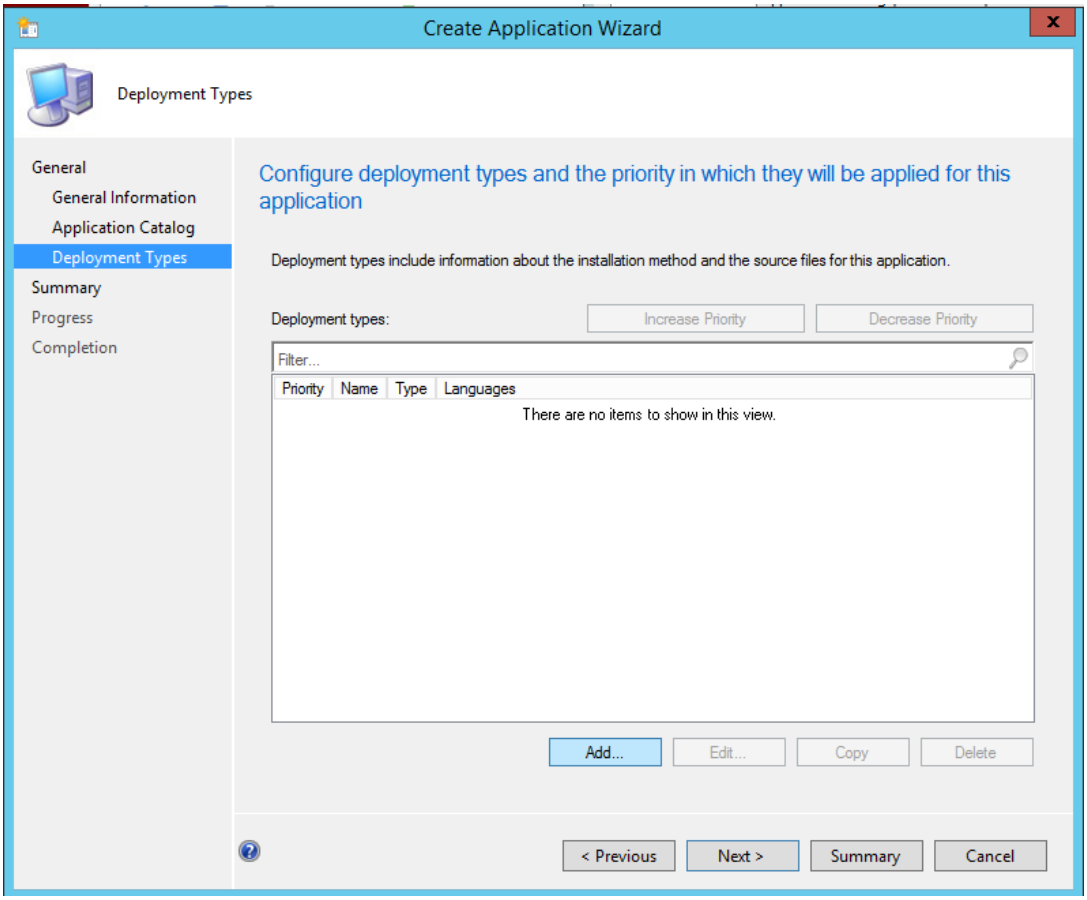
On the *Specify the Configuration Manager Application Catalog entry* screen, do the following:

- For packages requiring user interaction to install, ensure that the **Localized application name:** field includes a **- Manual** after the application name.
- Specify the application licensing type and any important installation notes in the **Localized description:** field.

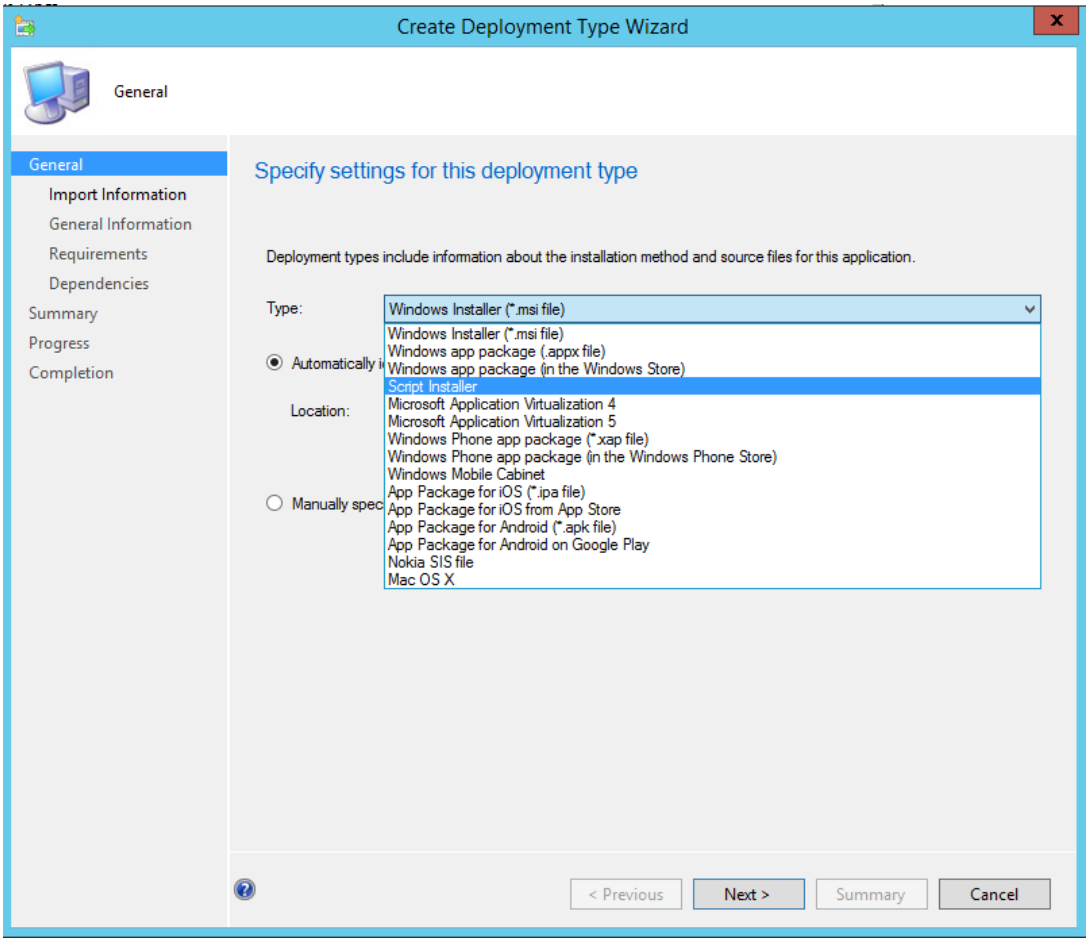
Click **Next** in the wizard.



On the *Configure deployment types and the priority in which they will be applied for this application* screen, click the **Add...** button to start the **Create Deployment Type Wizard**.



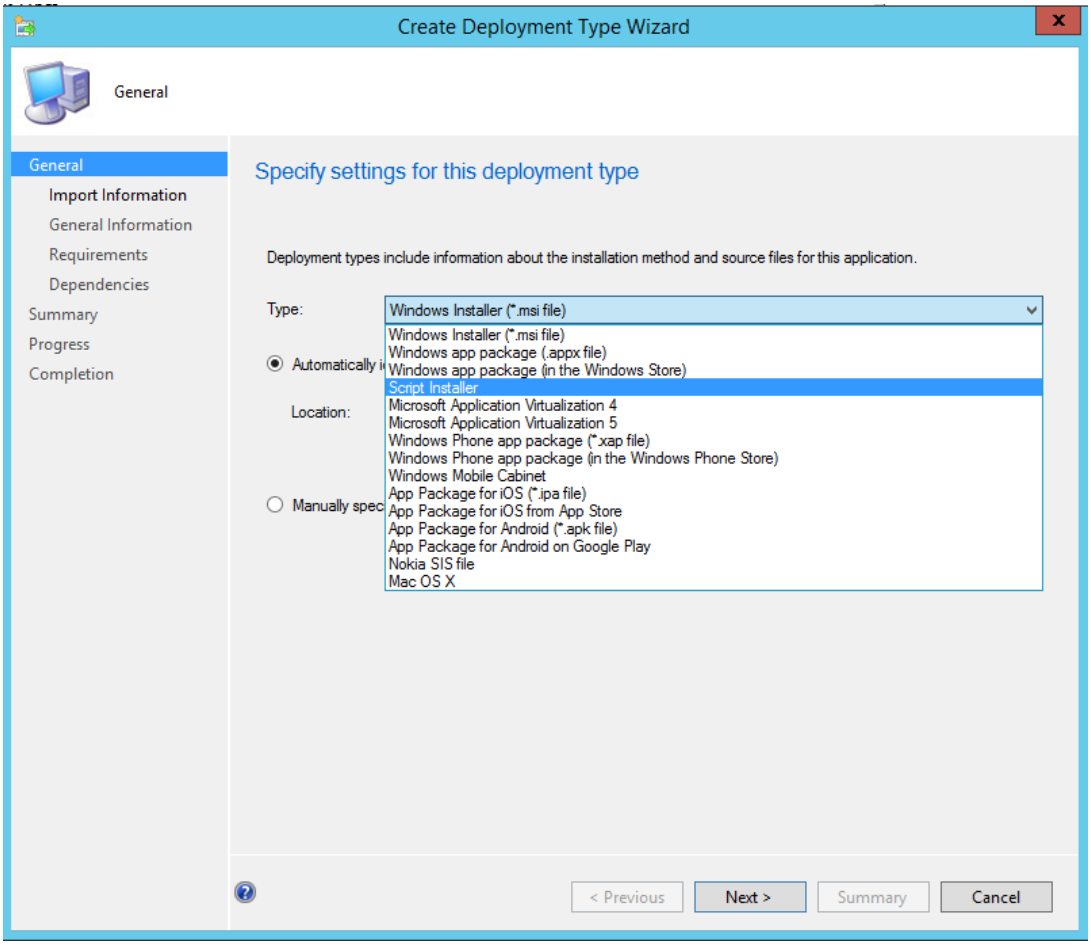
On the *Specify settings for this deployment type* screen in the **Create Deployment Type Wizard**, select **Script Installer** from the **Type:** drop-down list. The **Manually specify the deployment type information** button will automatically be selected. Click **Next** in the wizard.



On the *Specify general information for this deployment type* screen, do the following:

- Provide a meaningful name for the setup package in the **Name:** field.
- Specify the application licensing type and any important installation notes in the **Administrator comments:** field.

Click **Next** in the wizard.



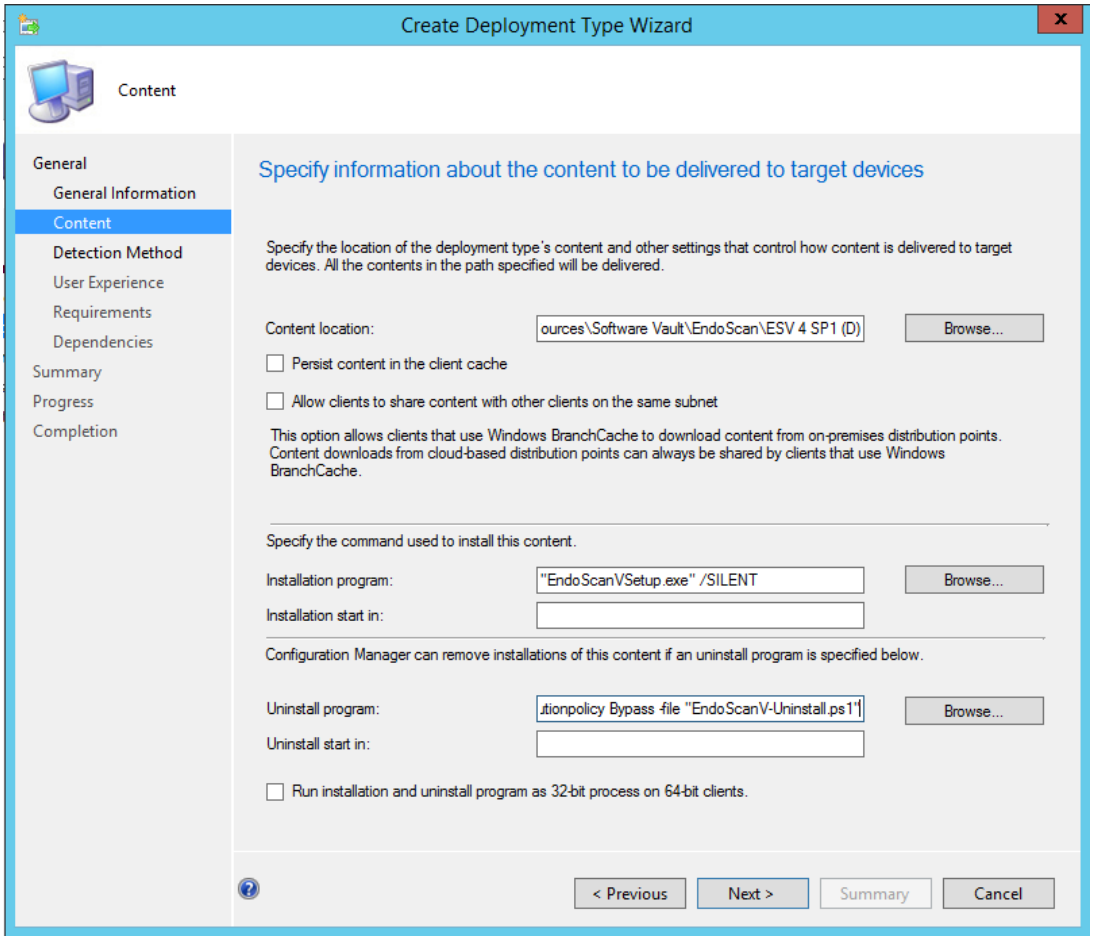
(Incorrect screenshot – duplicate of last screenshot)

On the *Specify information about the content to be delivered to target devices* screen, do the following:

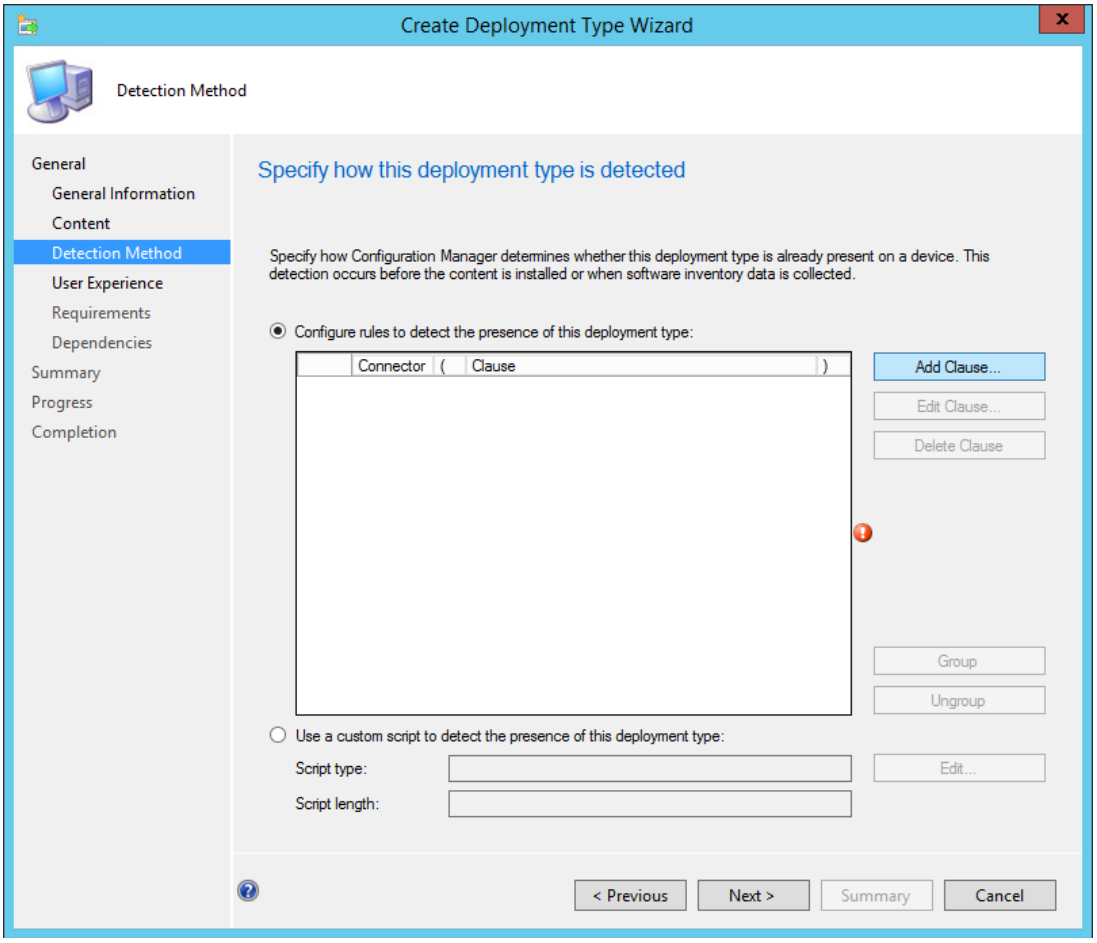
- Click the **Browse...** button for **Content location:** and browse to the location of the application's setup files on the **Software Vault** share.
- Disable the **Allow clients to share content with other clients on the same subnet** checkbox.
- Click the **Browse...** button for **Installation program:** and select the `.EXE` file or PowerShell script that starts the application installer.
- Click the **Browse...** button for **Uninstall program:** and select the PowerShell script that uninstalls the application and its components.

Note: Any PowerShell scripts called as the **Install program** or **Uninstall program** need to be in the following format to execute properly on the client system.

```
Powershell.exe –executionpolicy Bypass –file "FileName.ps1"
```



On the *Specify how this deployment type is detected* screen, click the **Add Clause...** button to open the **Detection Rule** page.

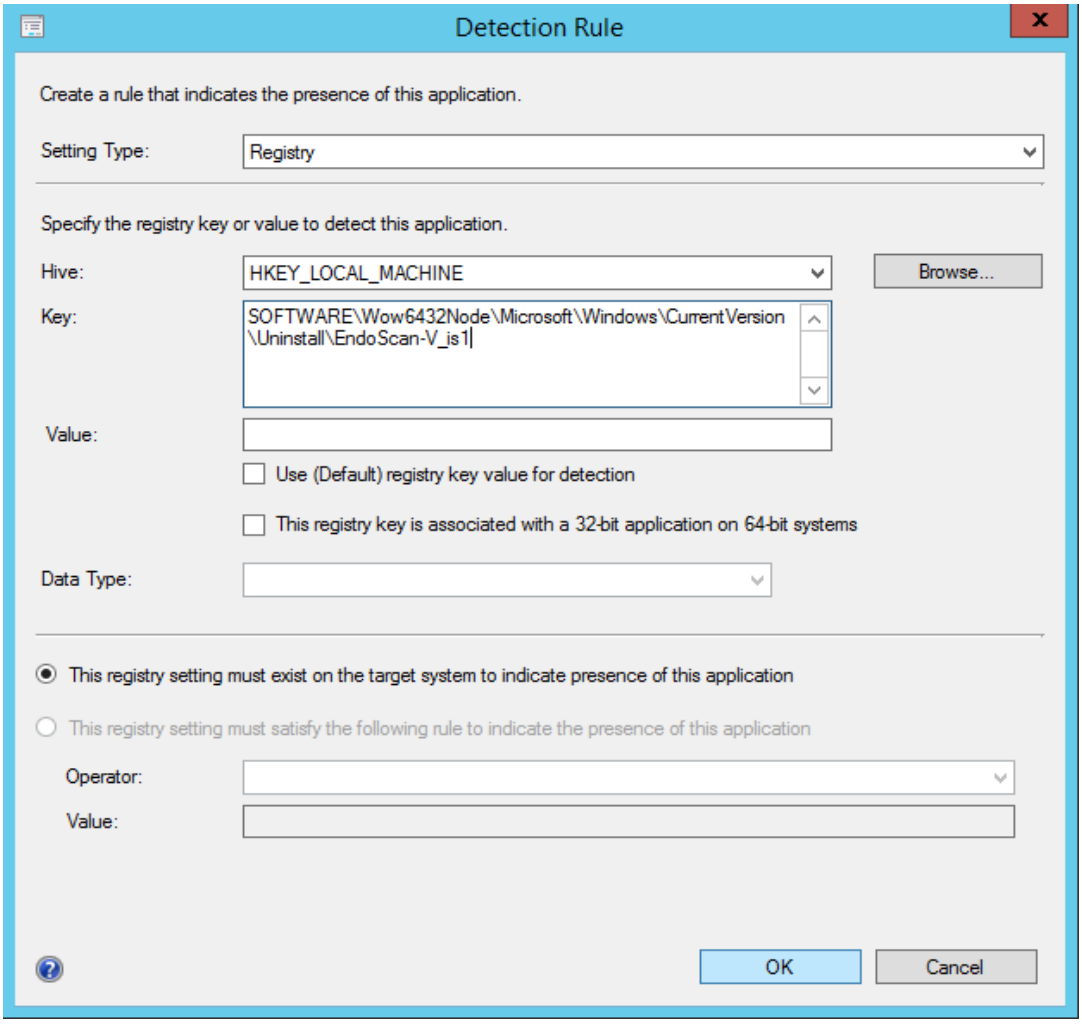


On the *Detection Rule* page, do the following:

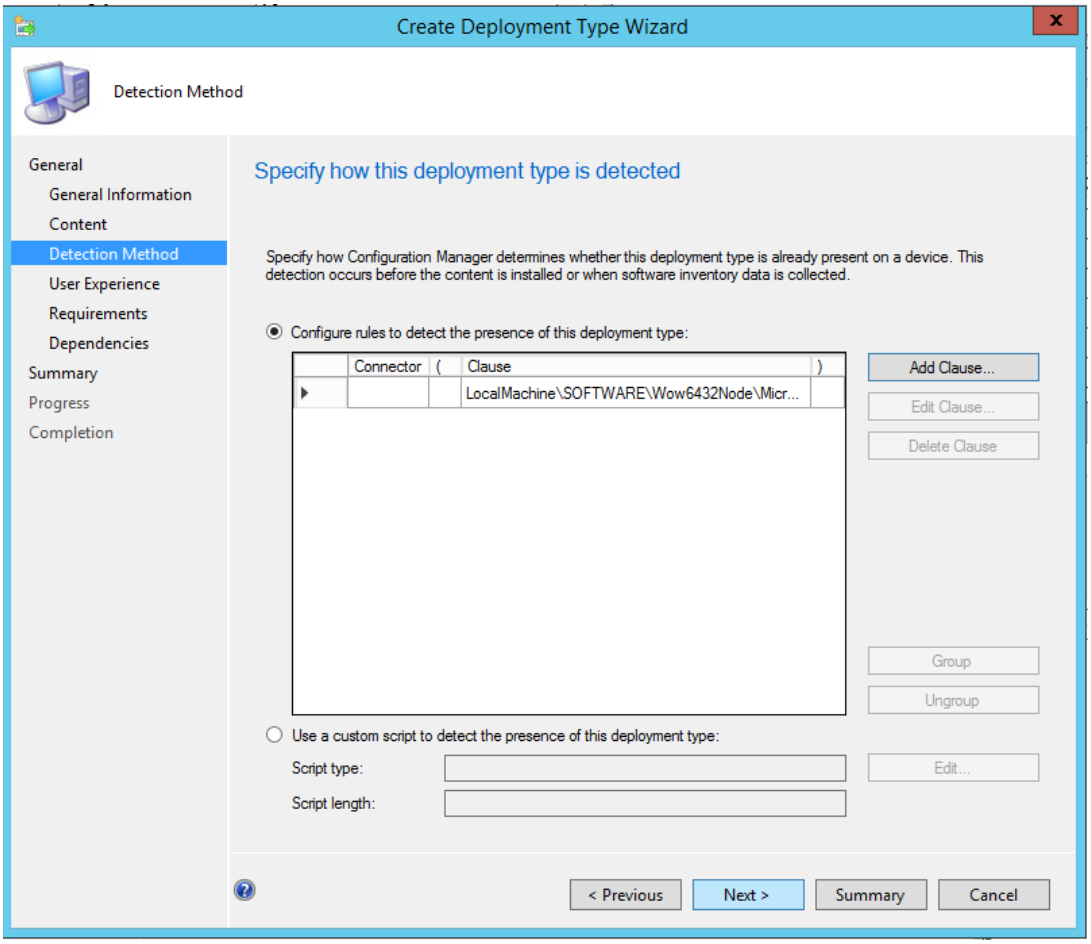
- From the package details obtained at the beginning, determine the type of **Setting Type** to be used for detecting this application.
  - **File System** -- used to detect the existence of a file or folder on the client system.
  - **Registry** -- looks for the existence of a registry key, or some registry property value for that key.
  - **Windows Installer** -- checks to see of an application with the specified GUID exists in the registry.
- For the purposes of this example, one would do the following:
  - Select **Registry** from the **Setting Type:** drop-down list.
  - Select **HKEY_LOCAL_MACHINE** from the **Hive:** drop-down list.
  - Paste the registry key location into the **Key:** text box.

    SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\EndoScan-V_is1

  - Click **OK** to exit the *Detection Rule* page.



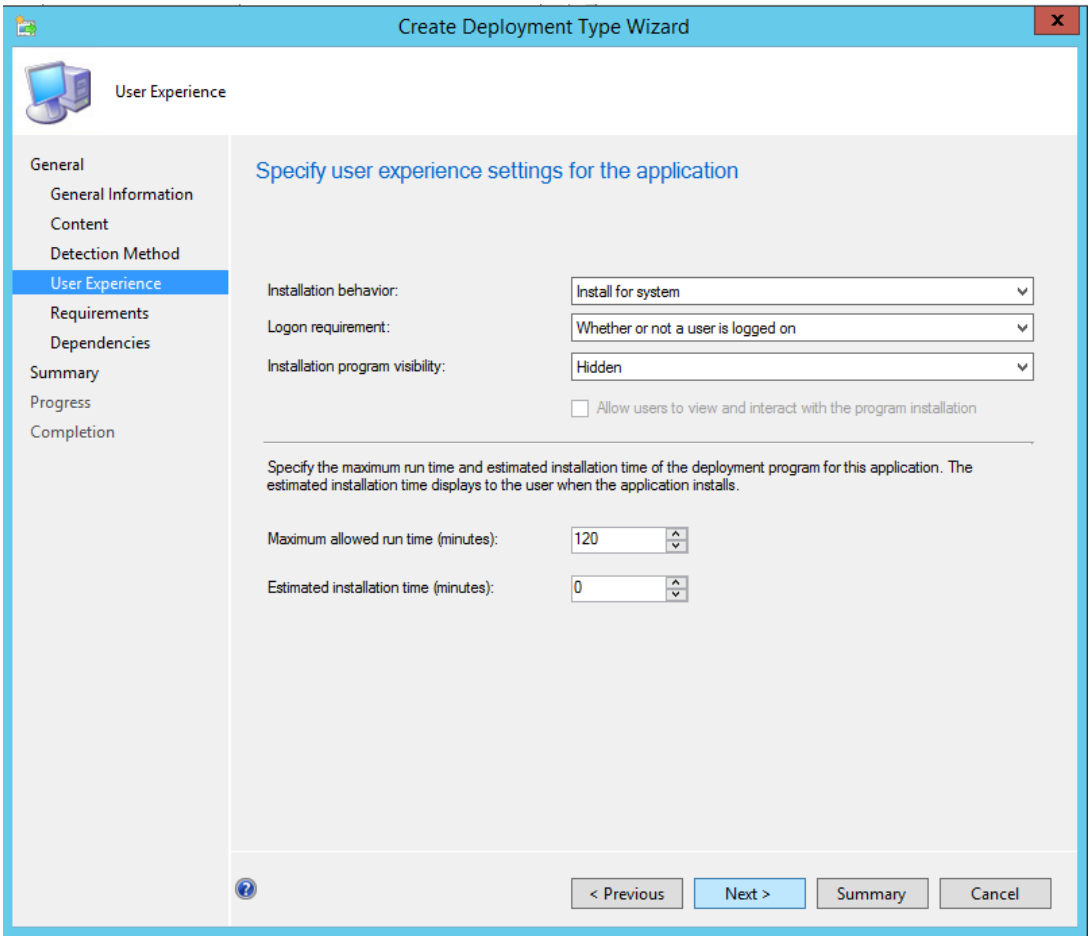
On the *Specify how this deployment type is detected* screen, click **Next** in the wizard.



On the *Specify user experience settings for the application* screen, select the user experience behavior from the drop-down lists for **Installation behavior**, **Logon requirement**, and **Installation program visibility**.

- For applications that DO NOT REQUIRE user interaction to install, set the following options:
  - Behavior: **Install for system**
  - Logon: **Whether or not a user is logged on**
  - Visibility: **Hidden**

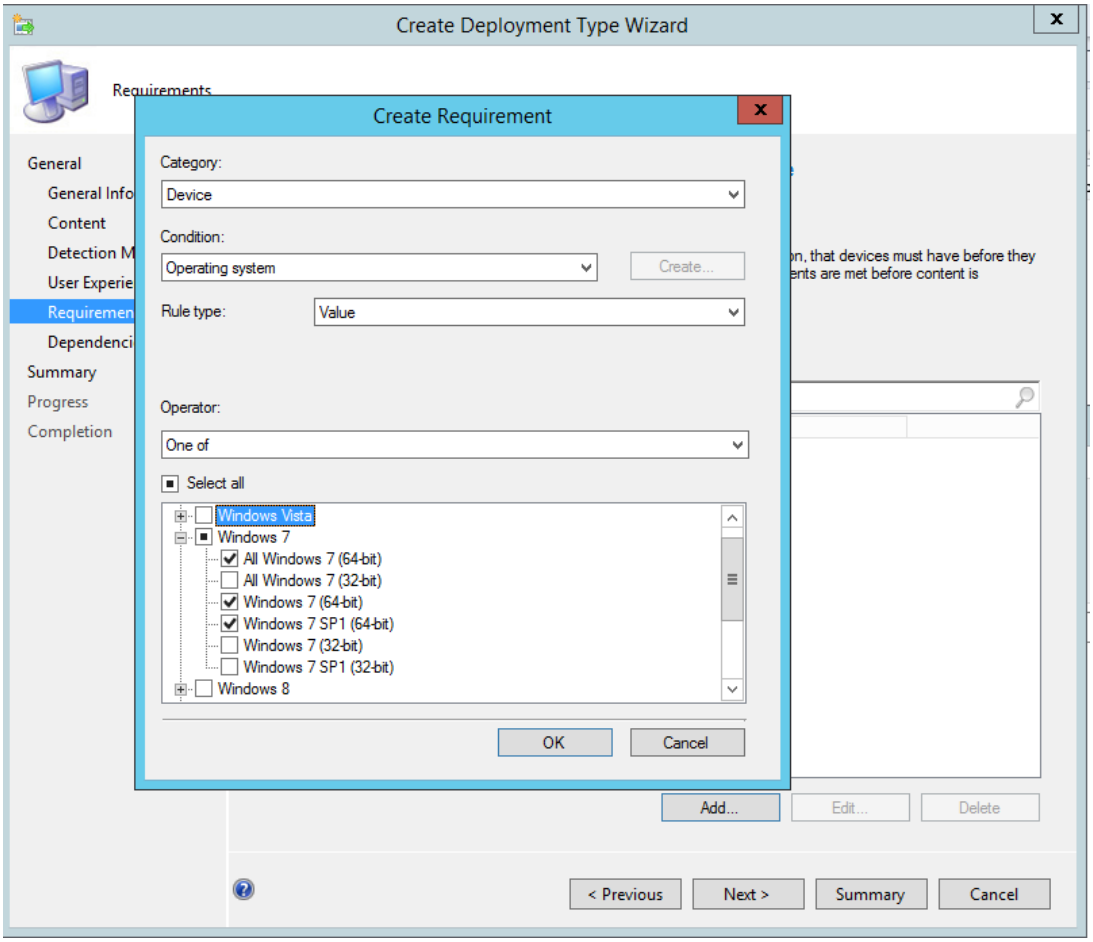
- For applications that REQUIRE user interaction to install, set the following options:
  - Behavior: **Install for user**
  - Logon: **Only when a user is logged on**
  - Visibility: **Normal**



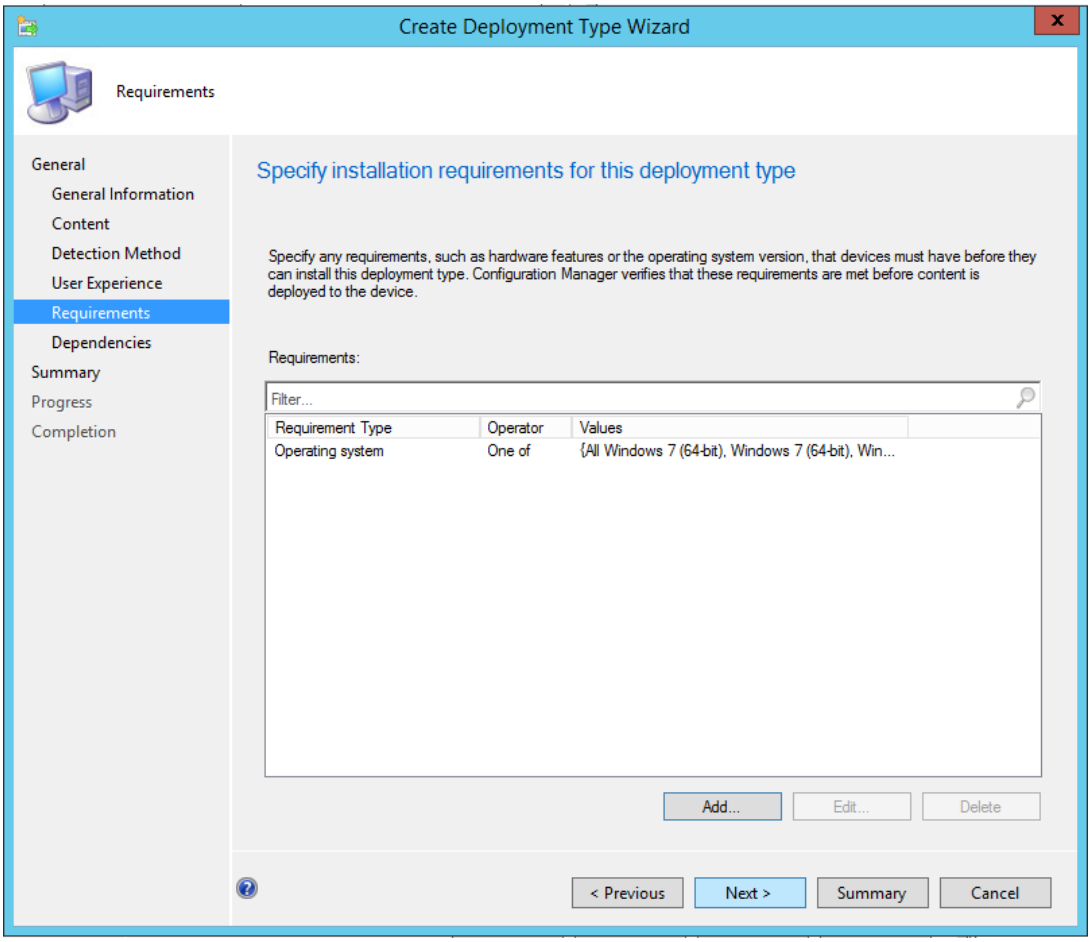
Note: If an application takes a long time to install, set the **Visibility** to **Normal** to make the installer visible so technicians can see that the installation is running without having to open the Task Manager to search for the process.

On the *Specify installation requirements for this deployment type* screen, click the **Add...** button to open the **Create Requirement** page to specify any hardware features or operation system versions that devices must have before application can be installed.

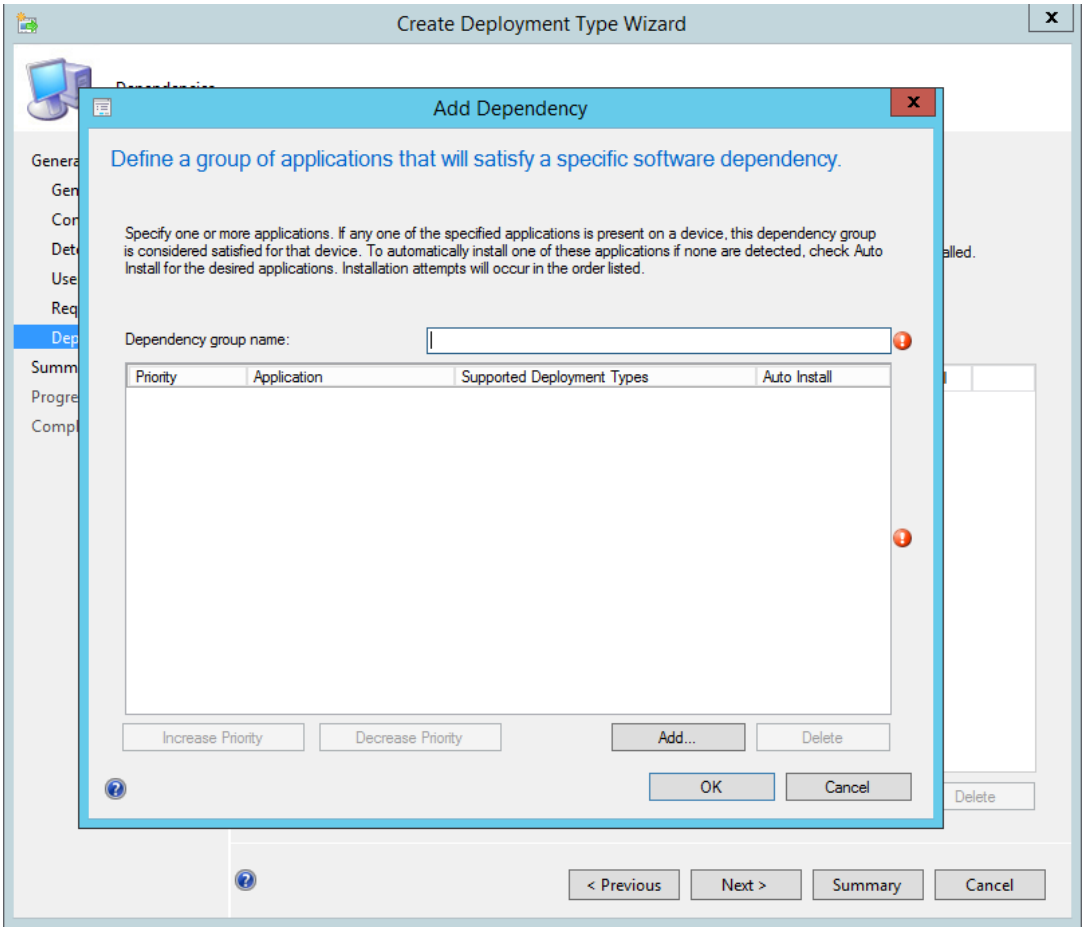
For this example, only **Windows 7 (64-bit)** systems will be able to install the application.



On the *Specify installation requirements for this deployment type* screen, click **Next** in the wizard.



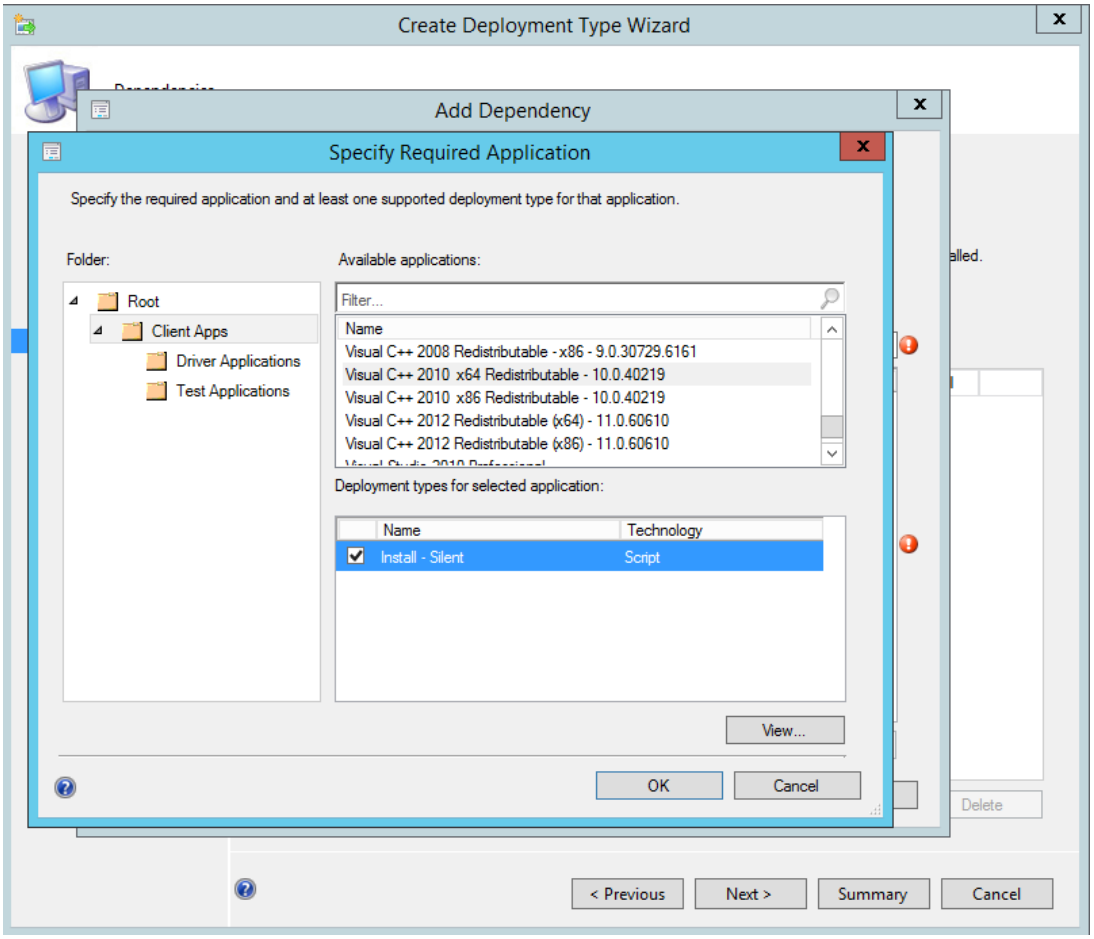
On the *Specify software dependencies for this deployment type* screen, click the **Add...** button to open the **Add Dependency** page to specify any other application packages that must be installed as prerequisites for this application package.



On the **Add Dependency** page, click the **Add...** button to choose any other application packages to install as prerequisites for installing this application package.

On the **Specify Required Application** page, do the following:

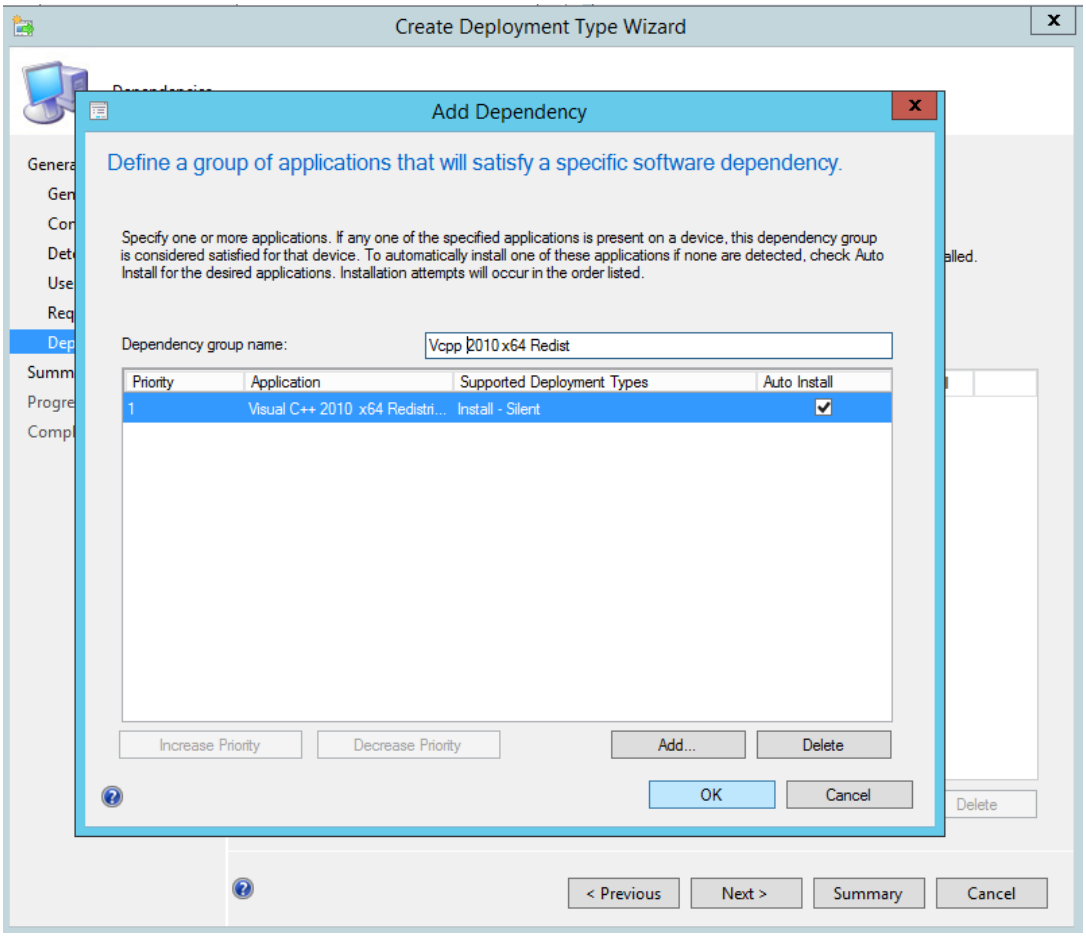
- Select **Client Apps** in the **Folder:** pane on the left.
- Find the prerequisite application package in the **Available applications:** list.
- Enable the checkbox next to the prerequisite application name in the **Deployment types for selected application:** lower, right-hand pane.
- Click **OK** to close the page.



Repeat the above steps for any other prerequisite applications packages required for installing this application package.

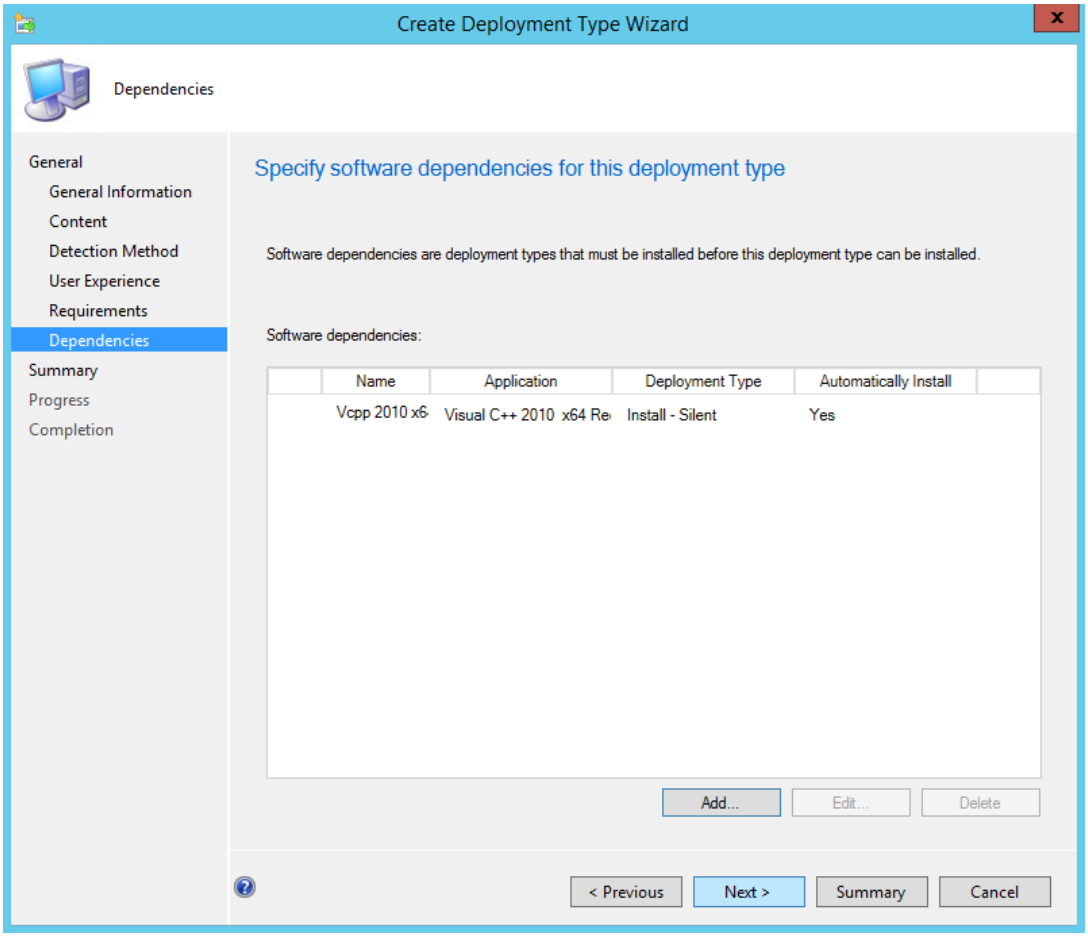
In the **Dependency group name:** field, provide a meaningful name for the prerequisite group.

Click **OK** to close the **Add Dependency** page.

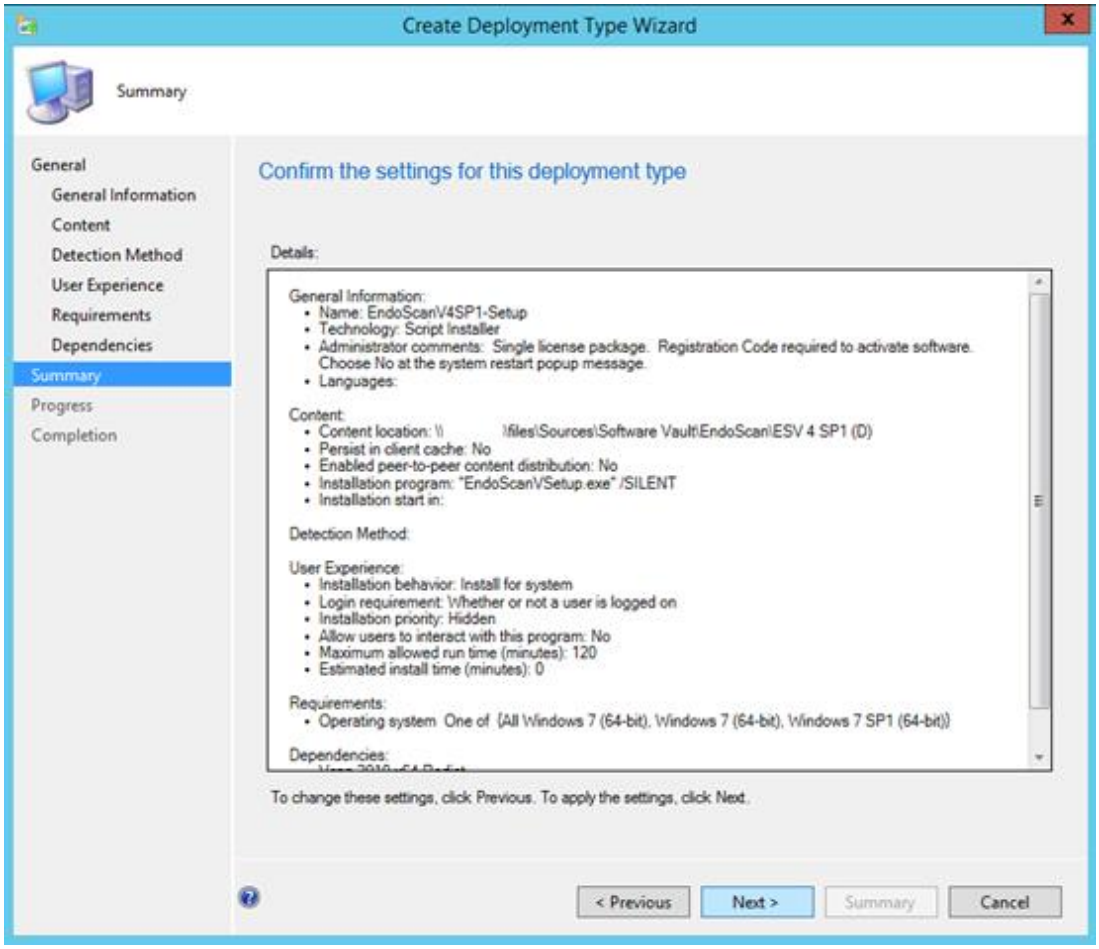


On the **Add Dependency** page, do the following:

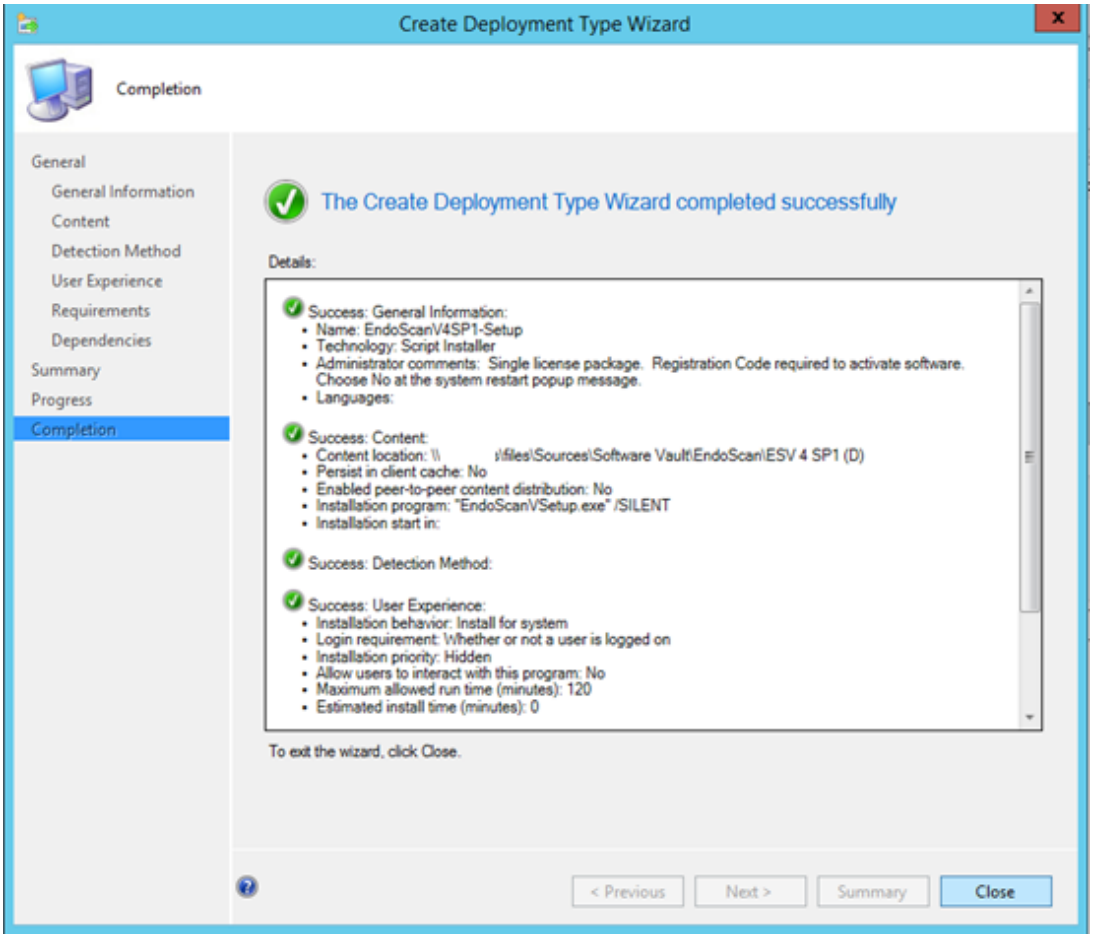
- Click the **Add...** button to choose more application packages to install as prerequisites for installing this application package.
- Or choose **Next** in the wizard.



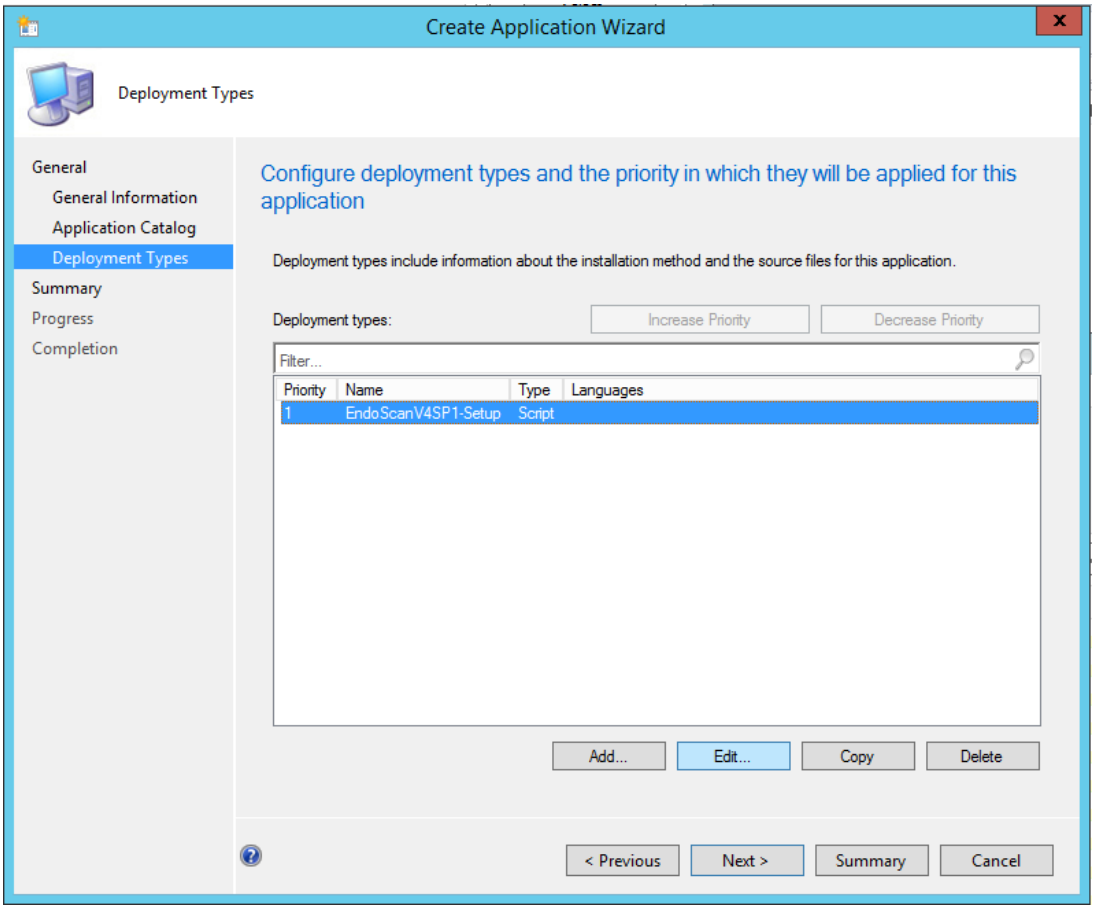
On the *Confirm the settings for this deployment type* screen, click **Next** in the wizard.



On the *Completion* screen, click **Close** to close the **Create Deployment Type Wizard**.

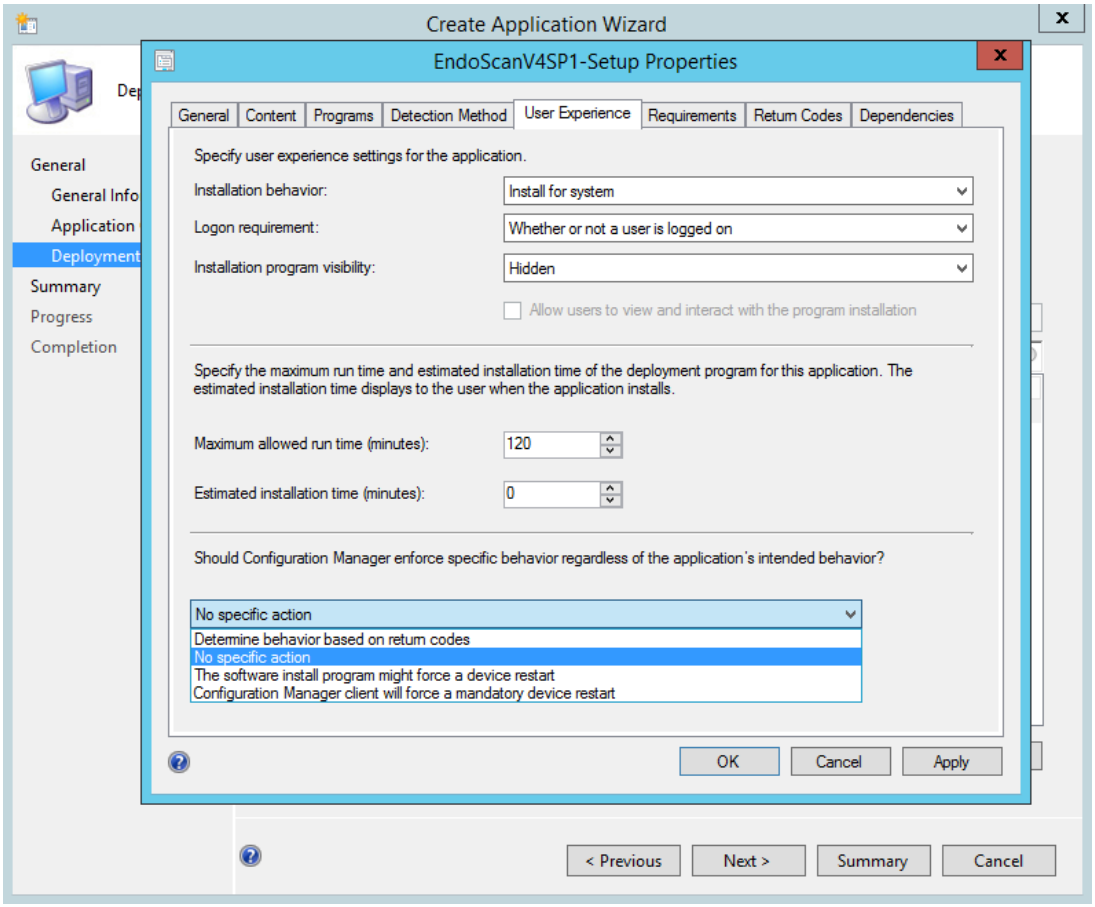


On the *Configure deployment types and the priority in which they will be applied for this application* screen, highlight the newly created deployment type and click the **Edit...** button.

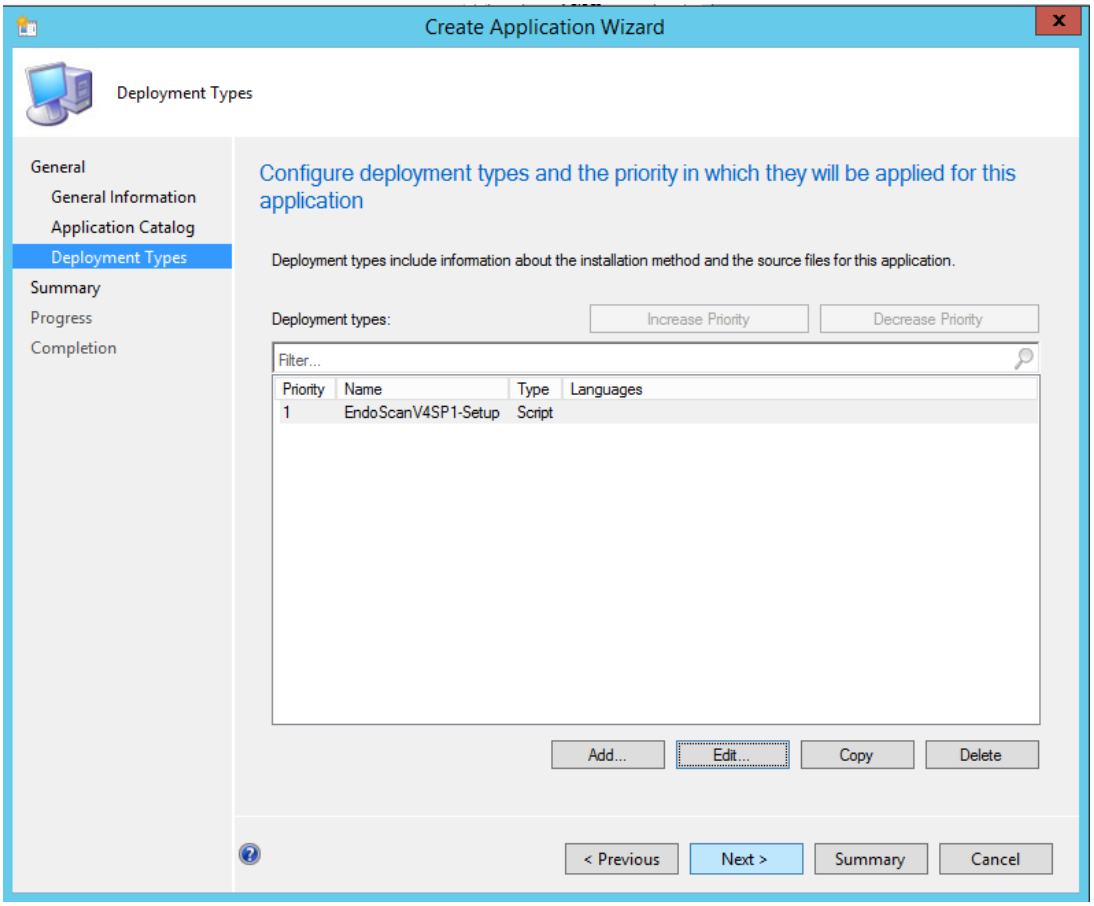


On the *Properties* page, click the **User Experience** tab and do the following:

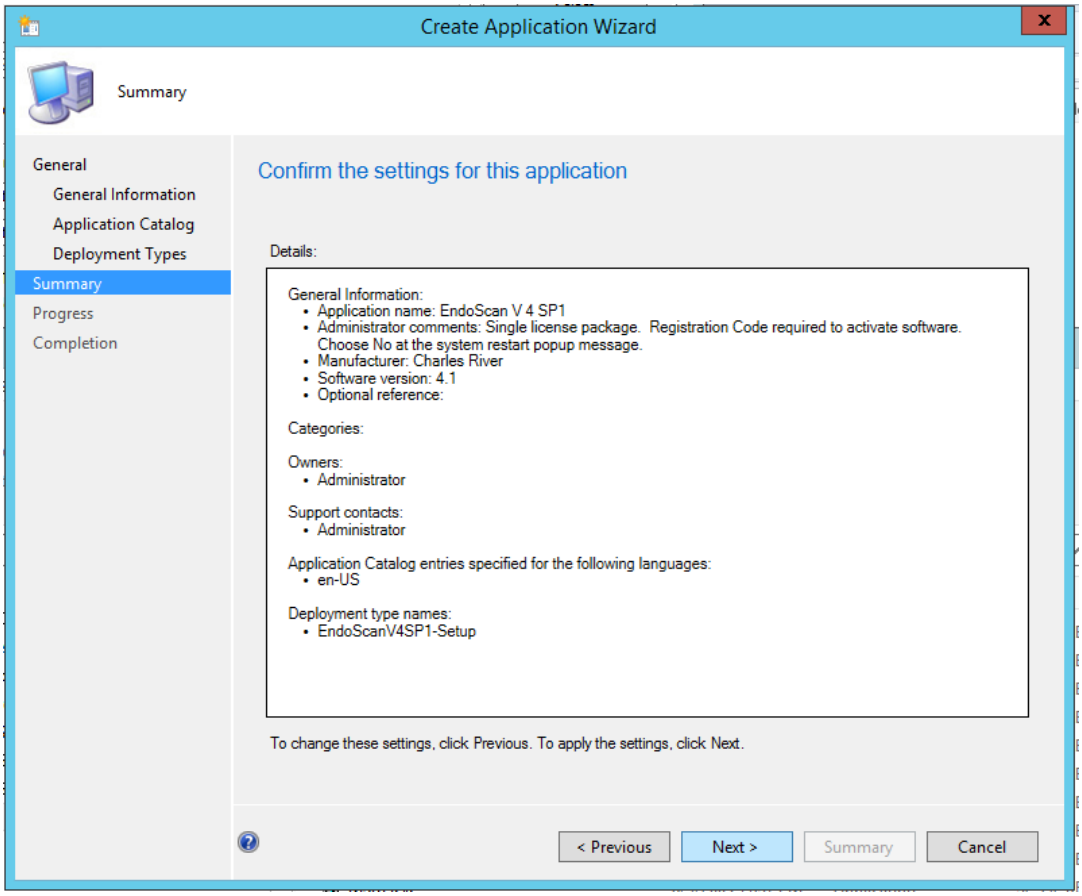
- Change the application behavior drop-down under **Should configuration Manager enforce specific behavior regardless of the application's intended behavior?** to **No specific action**.
- Click **OK** to exit the *Properties* page.



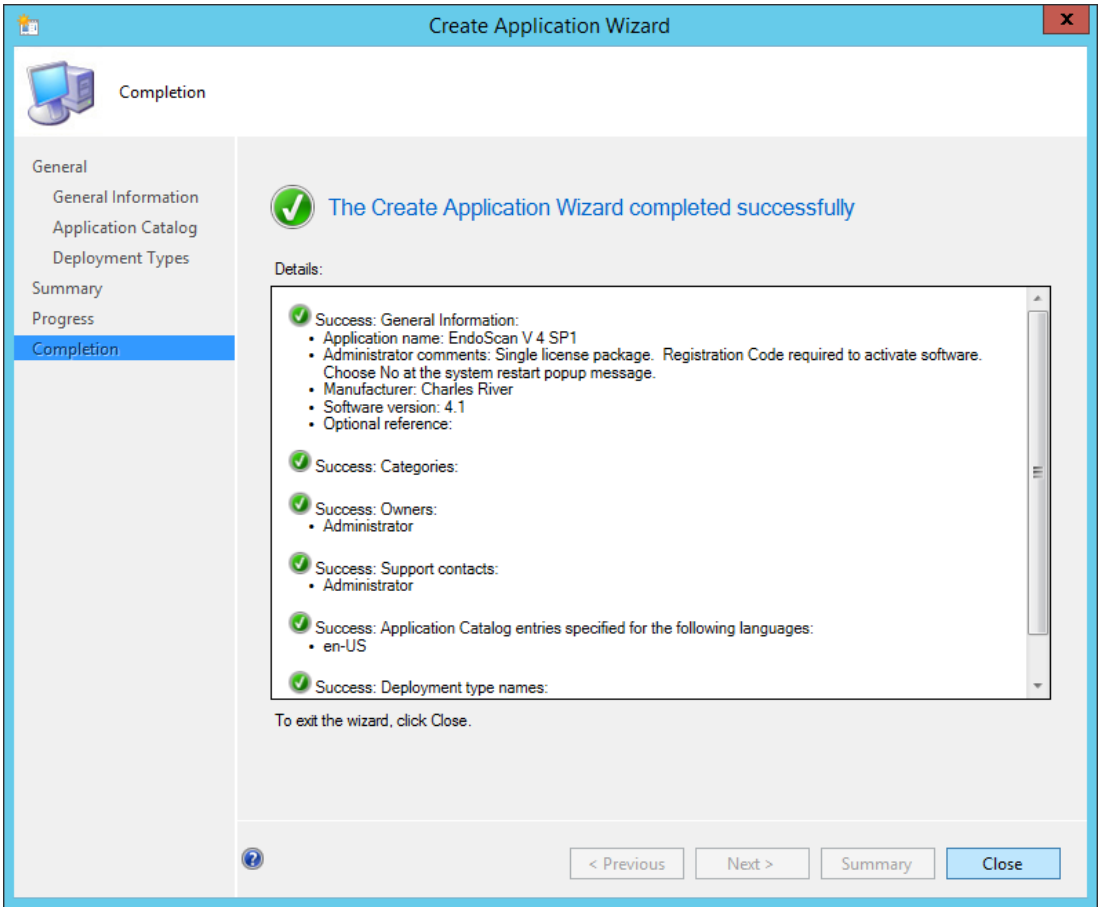
On the *Configure deployment types and the priority in which they will be applied for this application* screen, click **Next** in the wizard.



On the *Confirm the settings for this application* screen, click **Next** in the wizard.



On the *Completion* screen, click **Close** to exit the **Create Application Wizard**.

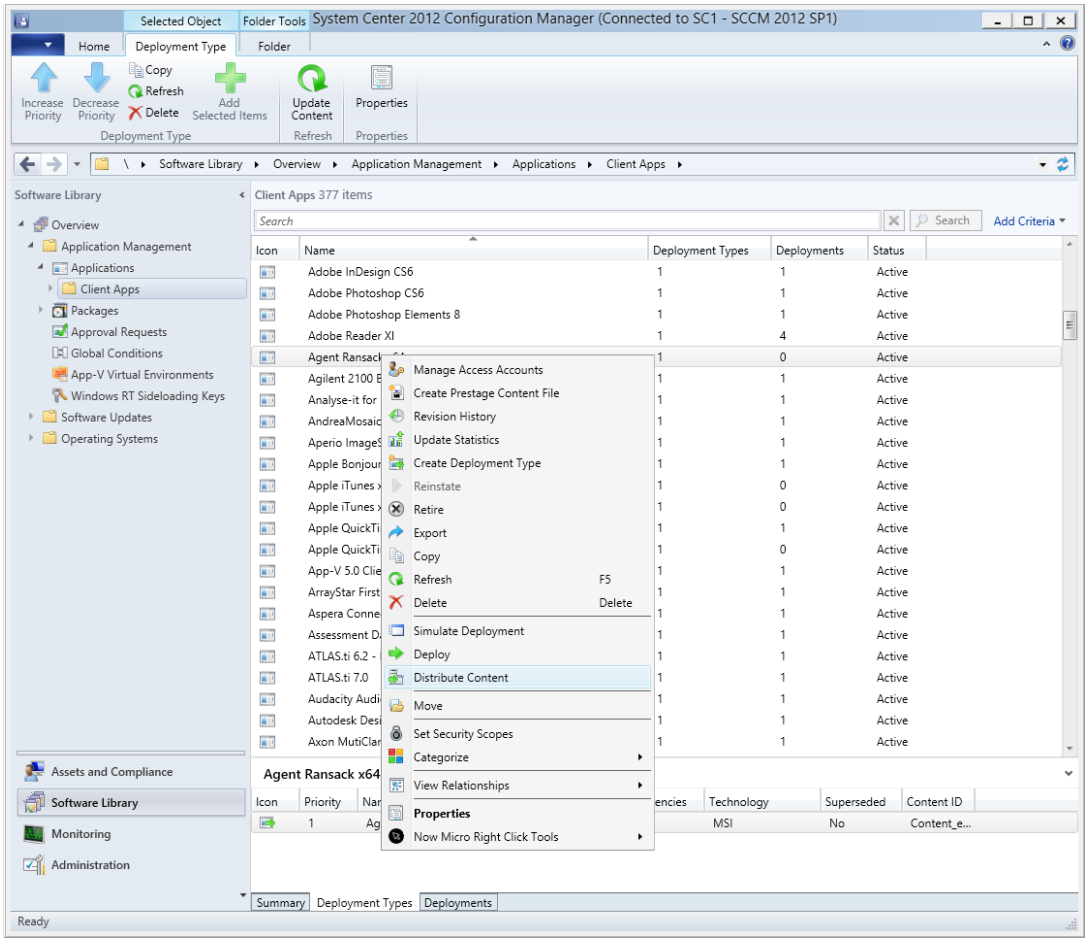


Find the application package in the SCCM **Client Apps** list and move on to the **4. How to Distribute Content** section.

4. How to Distribute Content

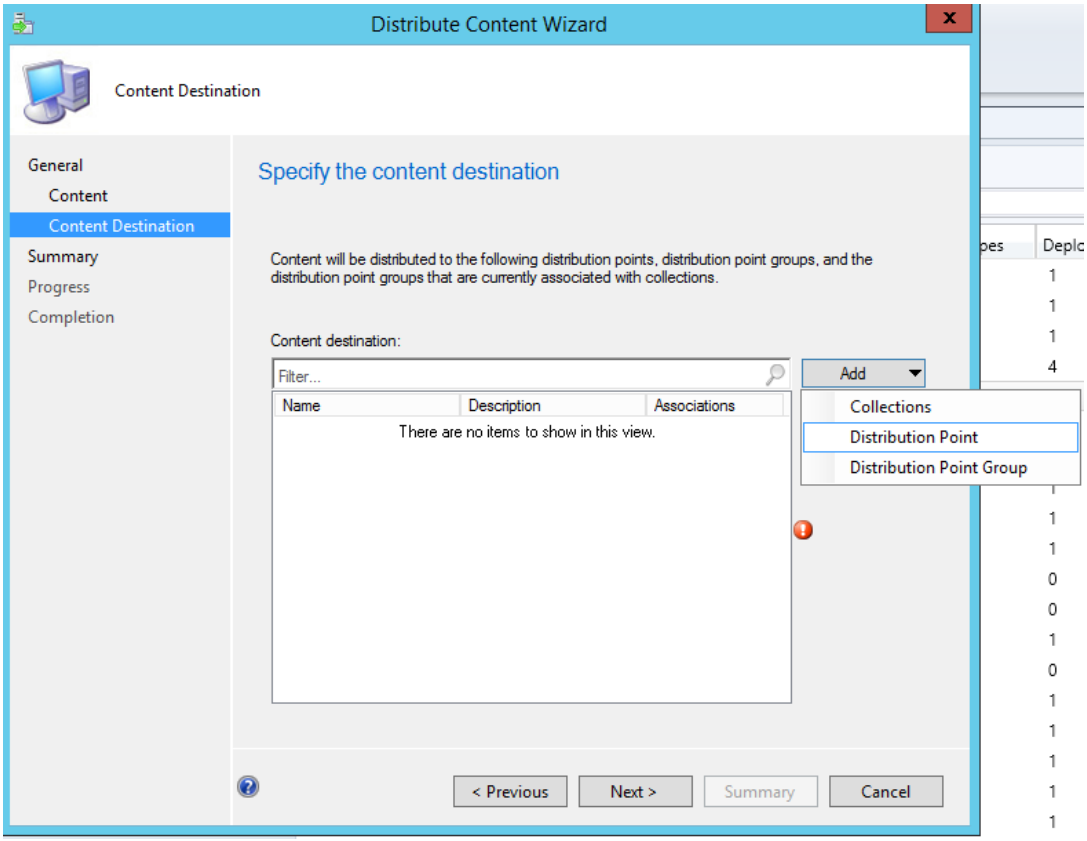
Close any open application files, like PowerShell uninstall scripts, in the Software Vault application folder before attempting to distribute content to the Distribution Servers. If a file in the Software Vault application folder is open, SCCM cannot distribute the content.

Find the application package in the SCCM **Client Apps** list, right-click the package name, and select **Distribute Content**.

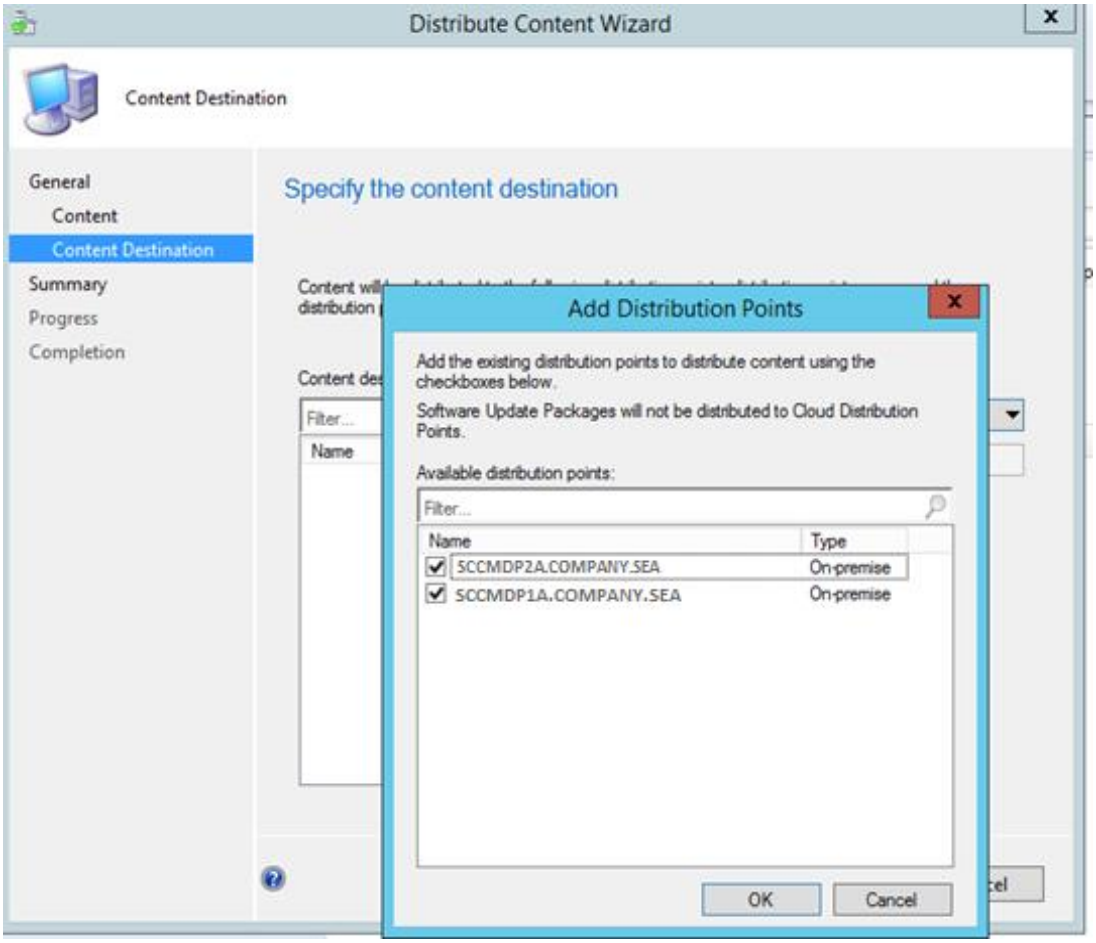


In the **Distribute Content Wizard**, do the following:

- Click **Next** past the first two screens of the wizard.
- Click the **Add** button on the *Specify the content destination* screen and select **Distribution Point**.



- Check both checkboxes on the **Add Distribution Points** page and click **OK** to close the window.

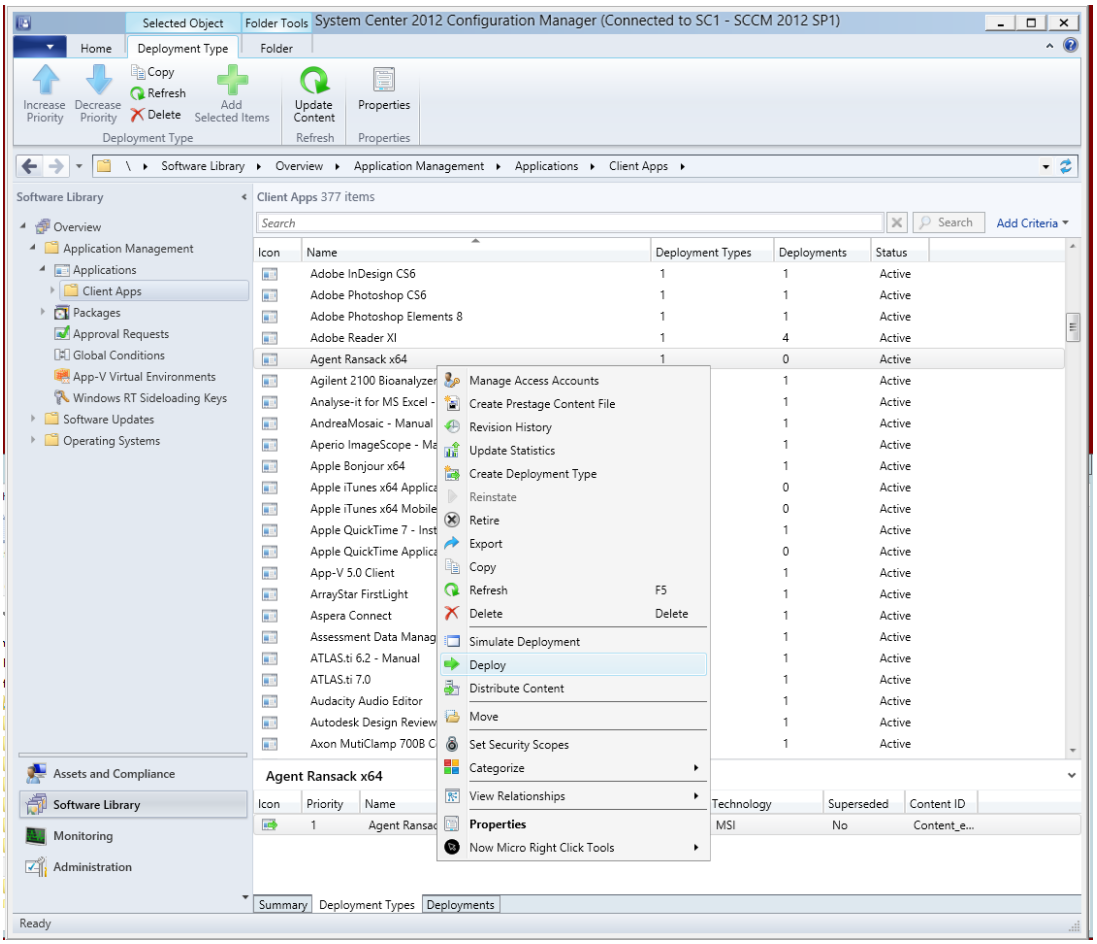


- Click **Next** twice and **Close** to exit the **Distribute Content Wizard**.

Find the application package in the SCCM Client Apps list and move on to the **5. How to Deploy a Package** section.

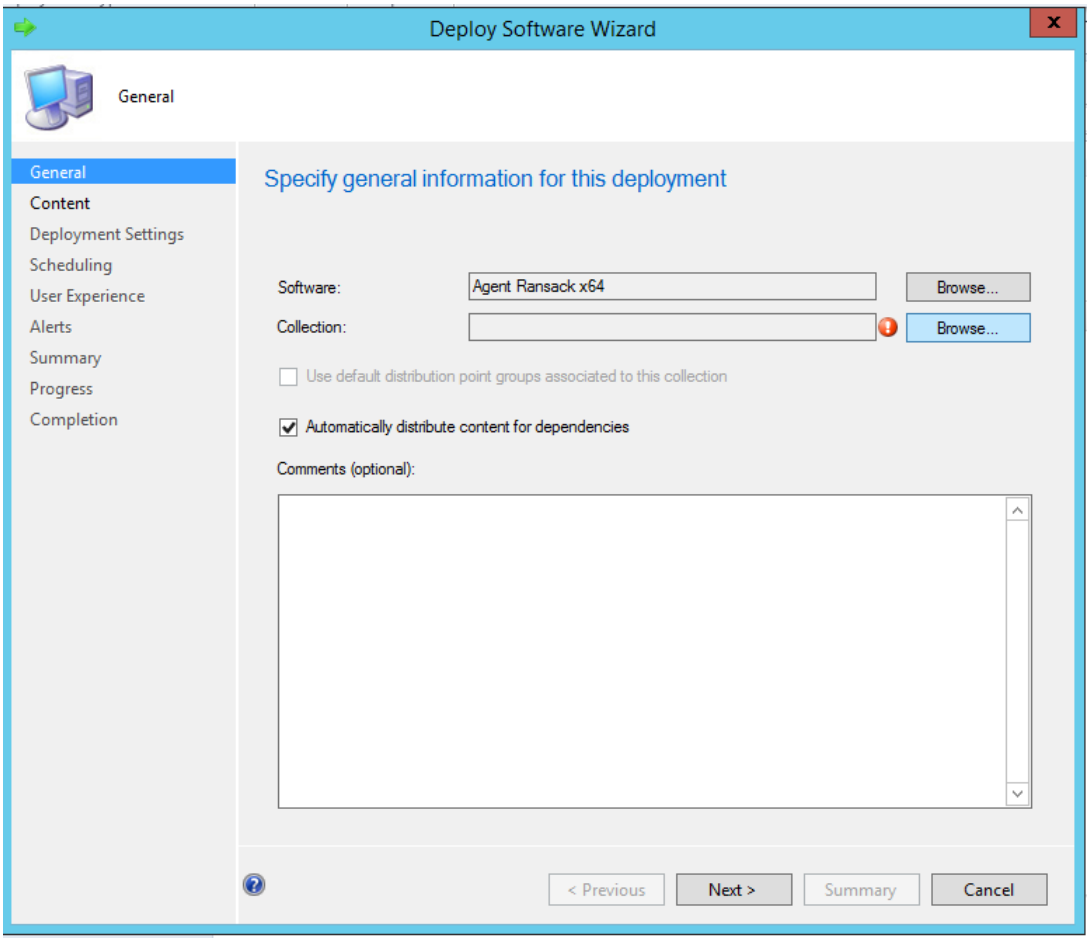
5. How to Deploy a Package

Find the application package in the SCCM **Client Apps** list, right-click the package name, and select **Deploy**.

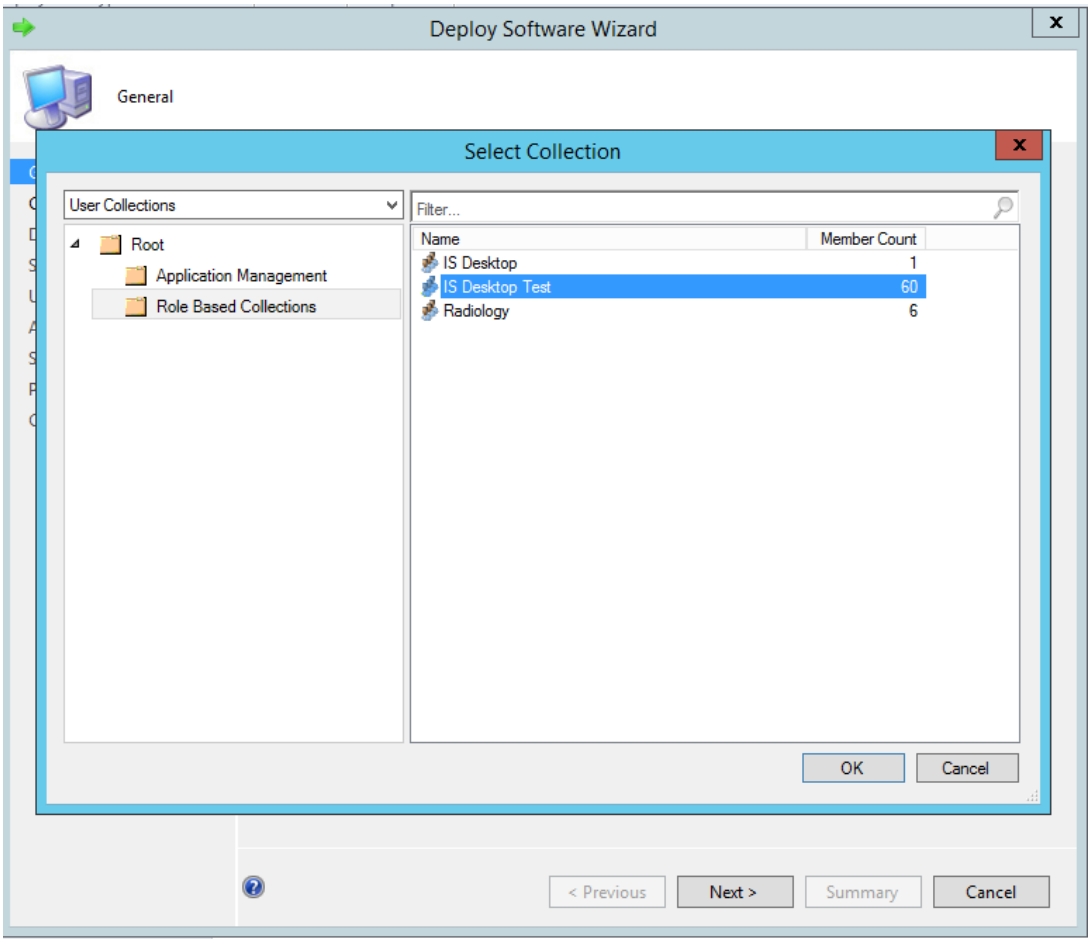


In the **Deploy Software Wizard**, do the following:

- On the *Specify general information for this deployment* screen, click the **Browse...** button next to **Collection:**

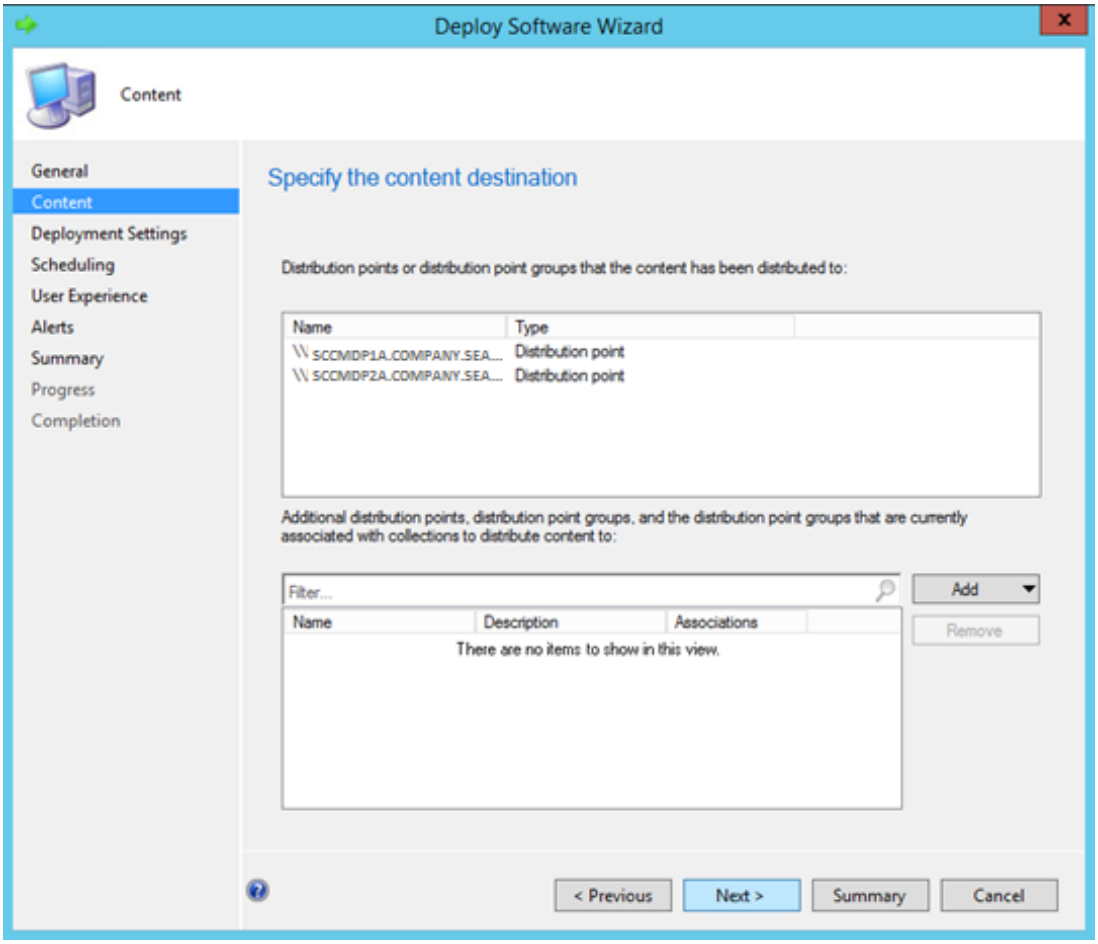


- On the **Select Collection** screen, select **Role Based Collections** under *User Collections*, then select **IS Desktop Test** in the right-hand pane, and click **OK** to close the window.



- Click **Next**.

- Wait for the *Specify the content destination* page to self-populate the distribution points list and then click **Next**.



- Accept all the defaults on the following pages by clicking **Next** and then click **Close** to exit the wizard.

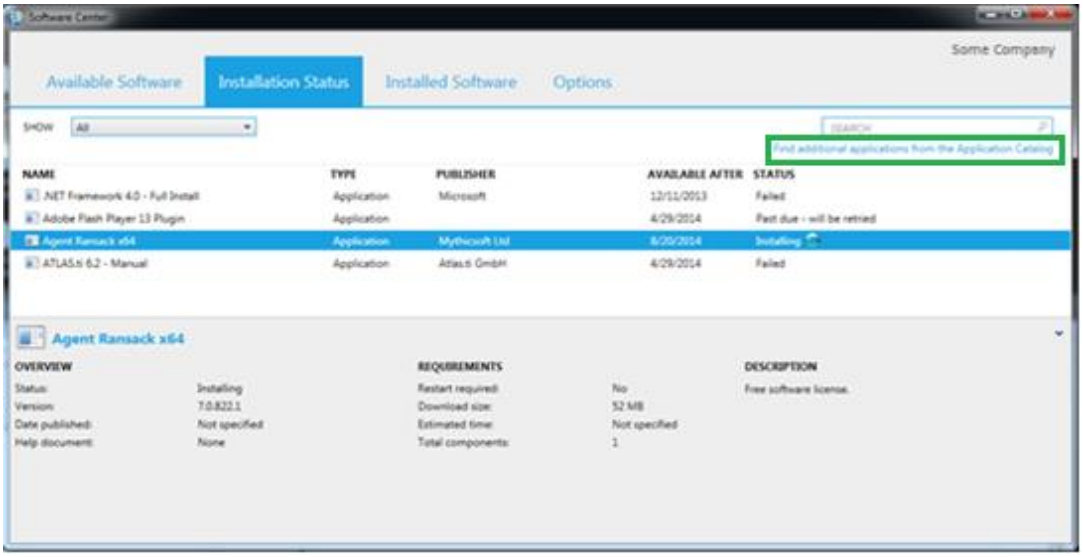
Be sure to wait 10-15 minutes for SCCM to complete processing before moving on to the next **6. Testing the Application Package** section.

6. Testing the Application Package

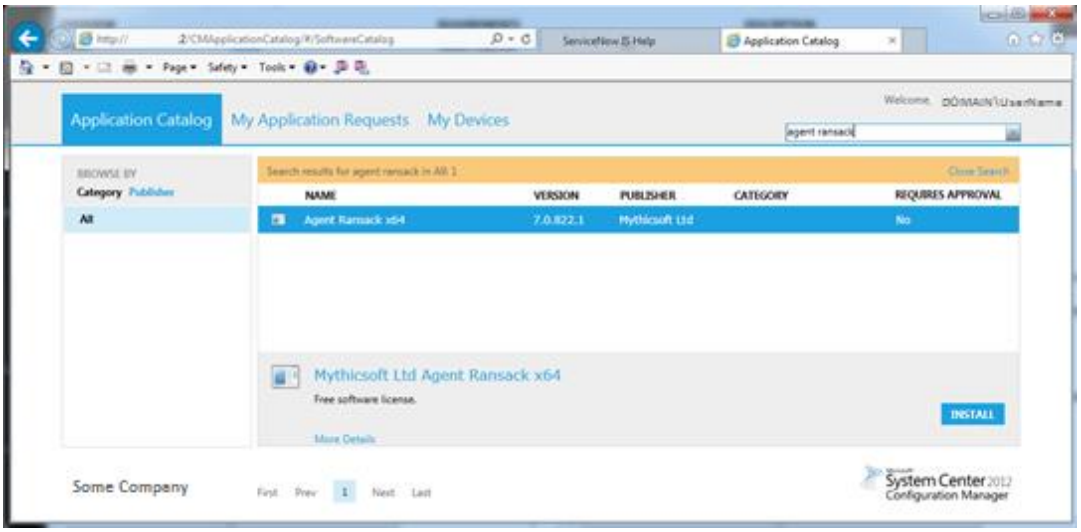
After waiting for the 10-15 minutes required for SCCM to copy the application contents from the Software Vault to the Distribution Points, test the SCCM application package install.

On a Windows 7 system, start **Software Center** which can be found in the **Start Menu** under *Microsoft System Center 2012*, *Configuration Manager*.

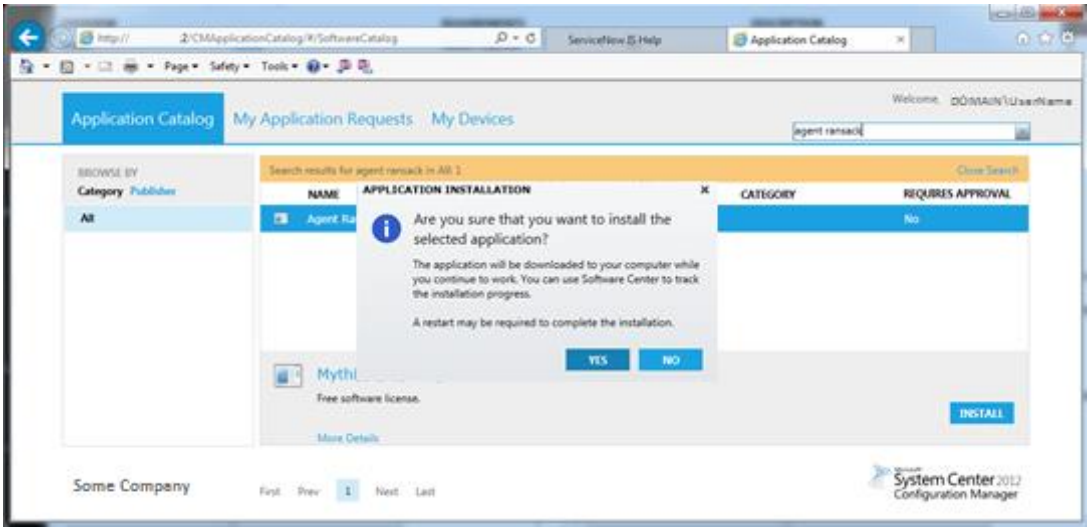
In **Software Center**, select the **Installation Status** tab, and click the *Find additional applications from the Application Catalog* link under the *SEARCH* text box.



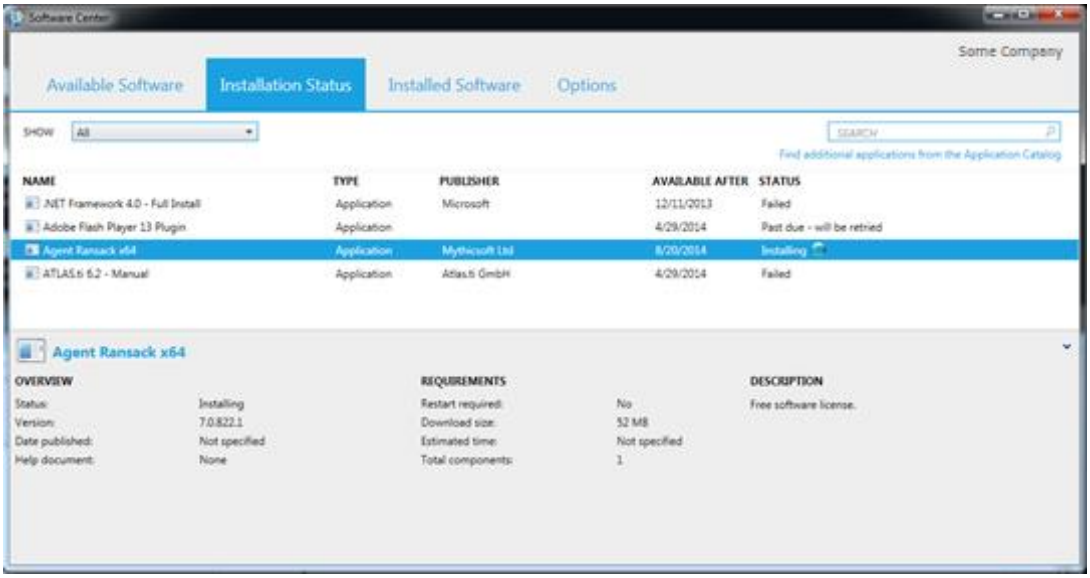
After Internet Explorer opens, type the application name in the *Search Application Catalog* text box and press **Enter**.



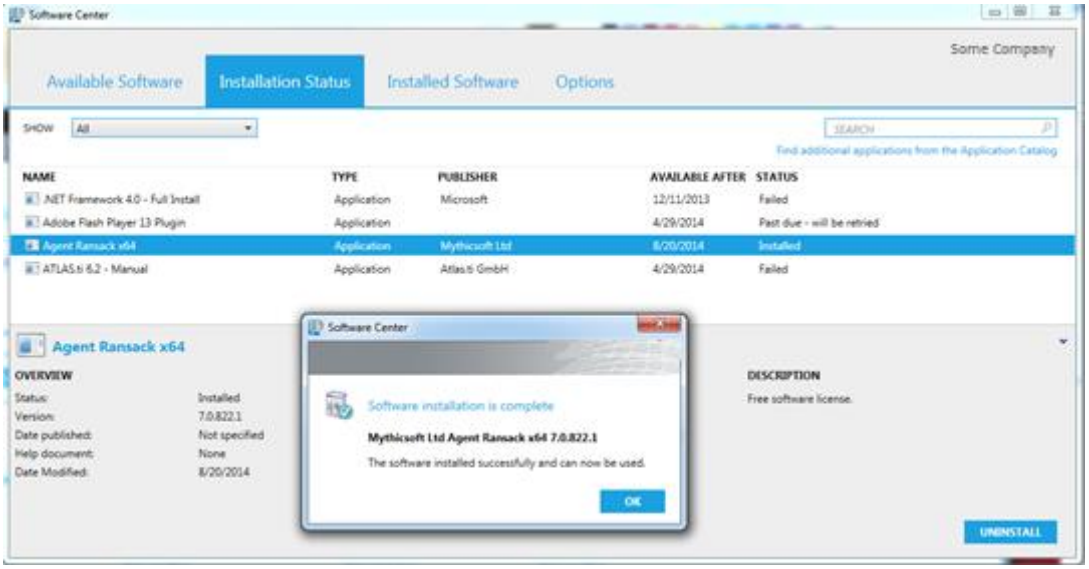
Click the **Install** button and click **Yes** on the *Application Installation* confirmation popup.



Switch back to the **Software Center** window to watch the application install.



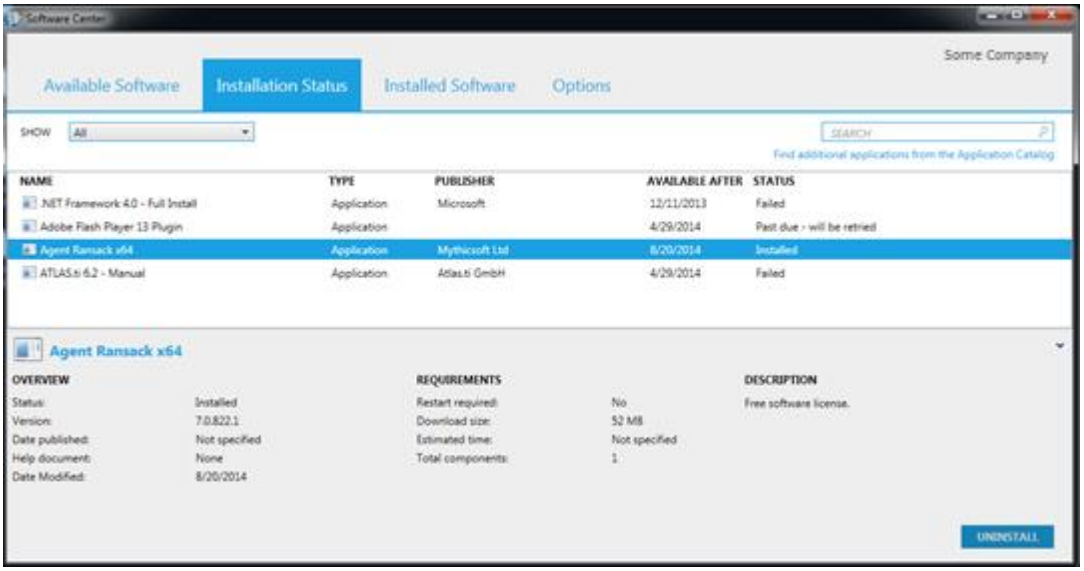
Click **OK** on the *Software installation is complete* popup message.



Start the application to ensure that it installed correctly.

If the application does create a shortcut in the **Start Menu** or on the user **Desktop**, then the installation method will need to be modified to a **Manual** method as the setup program requires user interaction to successfully create and set environmental variables.

In the **Software Center** window, select the application name and click the **Uninstall** button to ensure that the application is successfully removed from the local system.



The application should drop off the installed software list and a popup message indicating a **Removal Complete** should appear.

Install or uninstall errors need to be investigated to determine how the SCCM package needs to be modified.

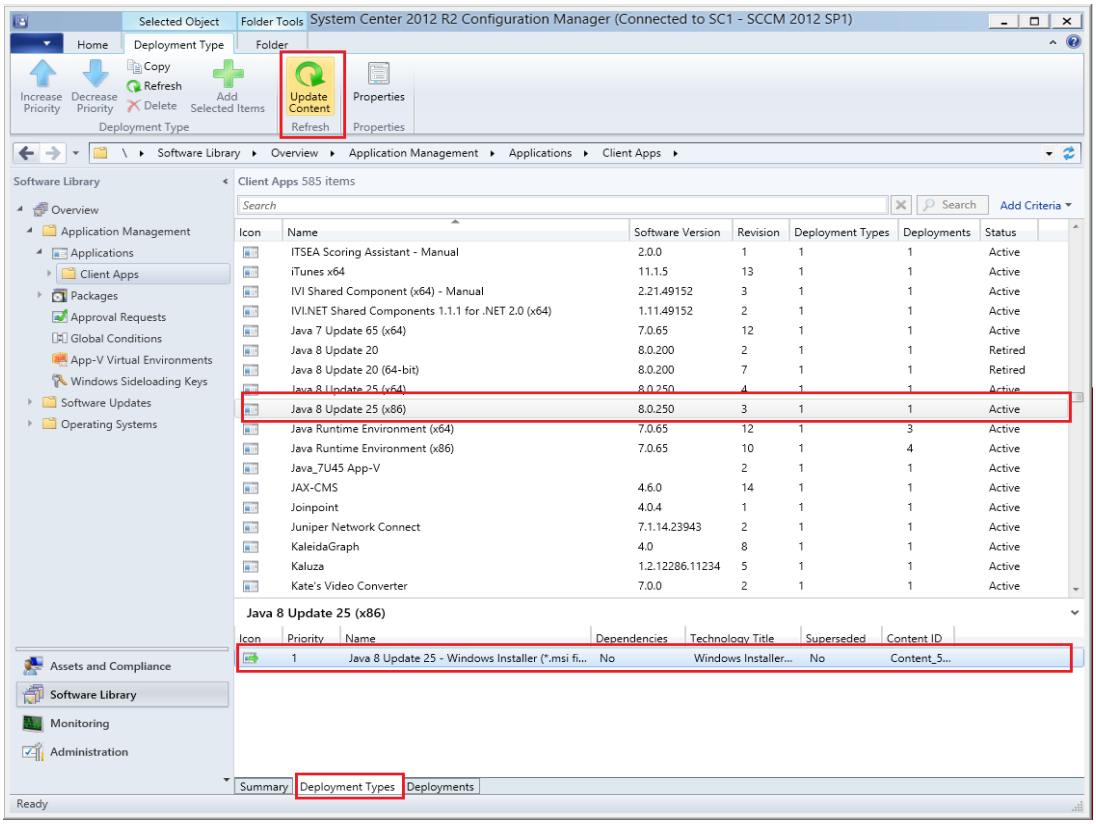
7. Re-Testing the SCCM Application Package

After modifying a package's settings or updating a file on the Software Vault, the package will need its content updated on the Distribution Servers. Then, before attempting to install through Software Center, the Configuration Manager cache needs to be cleared on the application packaging desktop test system.

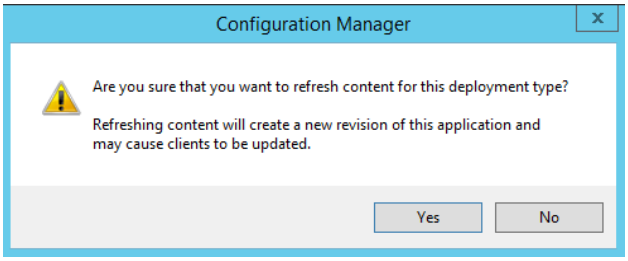
Updating Content on the Distribution Server

On the SCCM Configuration Manager console, do the following:

- Select the **Software Library**, expand **Applications**, and then expand **Client Apps.**
- Find the application package in the **Client Apps** list and select the application package.
- Select the **Deployment Types** tab at the bottom of the console window and select the installer name in the lower pane.
- Click the **Update Content** button below the menu bar at the top of the console window.



- Click **Yes** on the refresh content confirmation popup message.

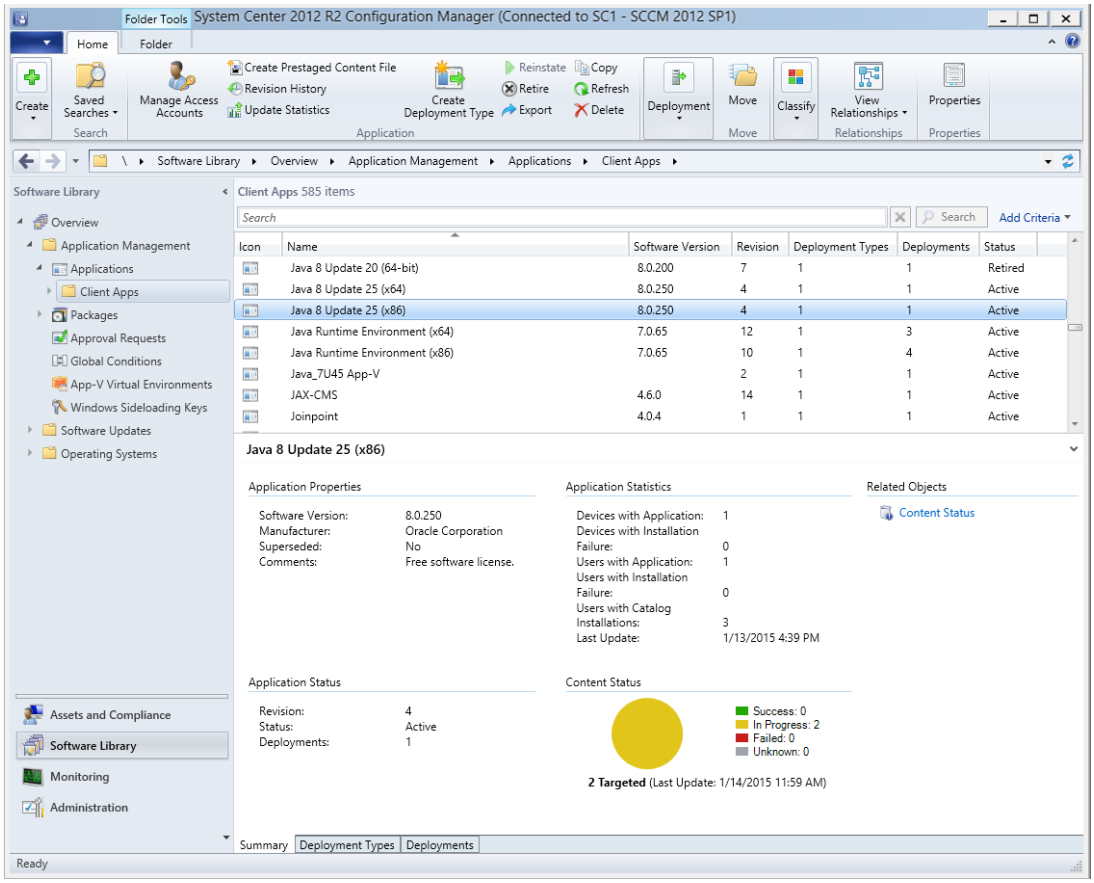


Verifying the Content Update on the Distribution Server

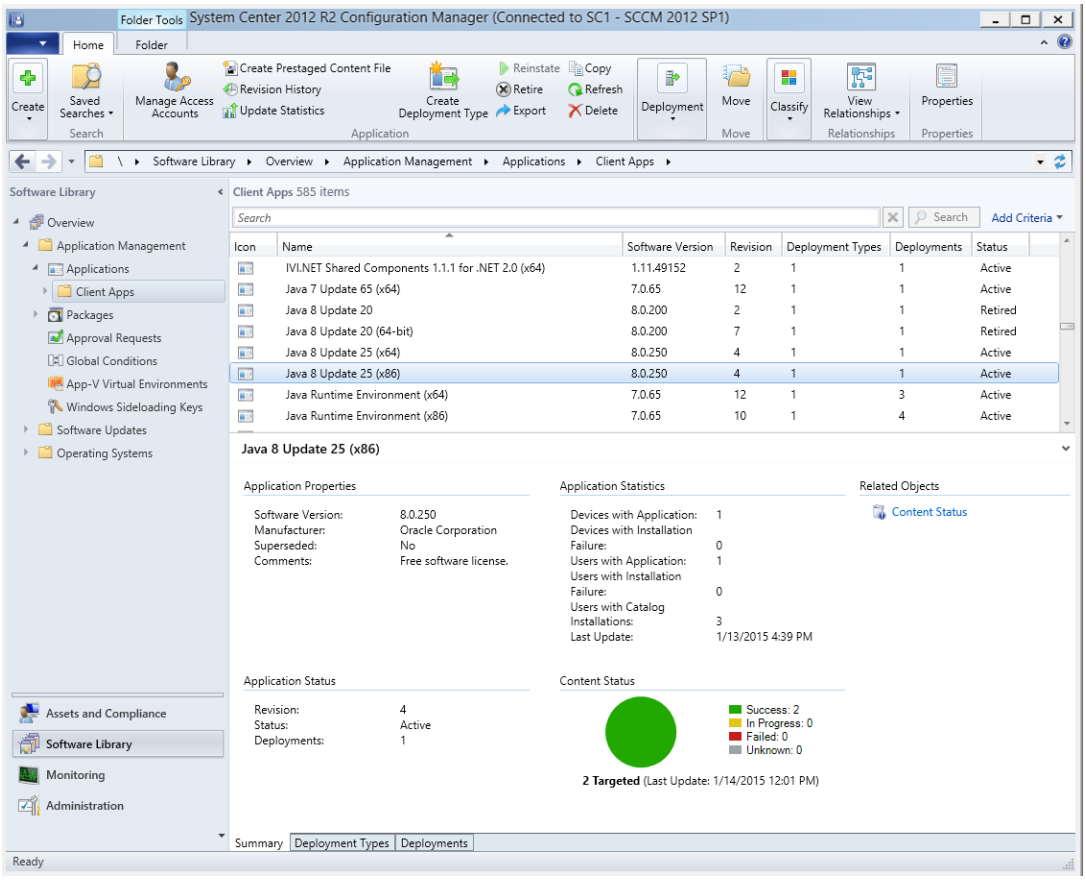
On the SCCM Configuration Manager console, do the following:

- Verify that the package is updating by selecting the **Summary** tab at the bottom of the console window and scrolling down to the **Content Status** section to view the content update status.
- Color key:
  - Green indicates a successful content update.
  - Yellow means the content update is in progress.
  - Red indicates a failed content update that needs to be investigated.
    - Make sure there are no open files in the application's **Software Vault** folder (e.g., an uninstall script open for editing in PowerShell or a ReadMe file in Notepad).

Java 8 Update 25 (x86) - Content Update in progress...



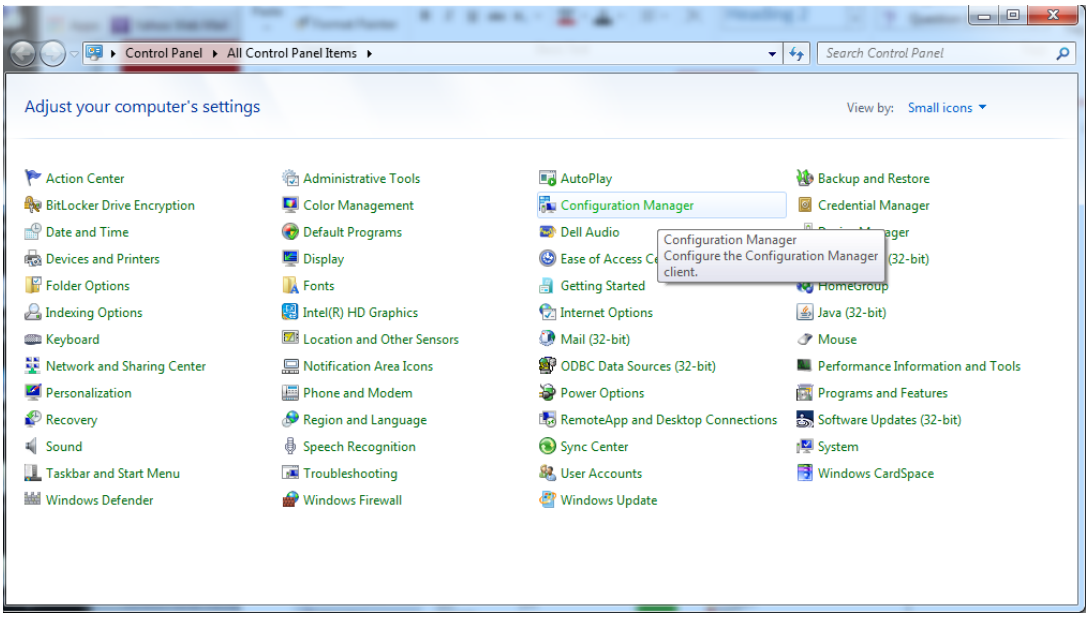
Java 8 Update 25 (x86) - Content Update success!



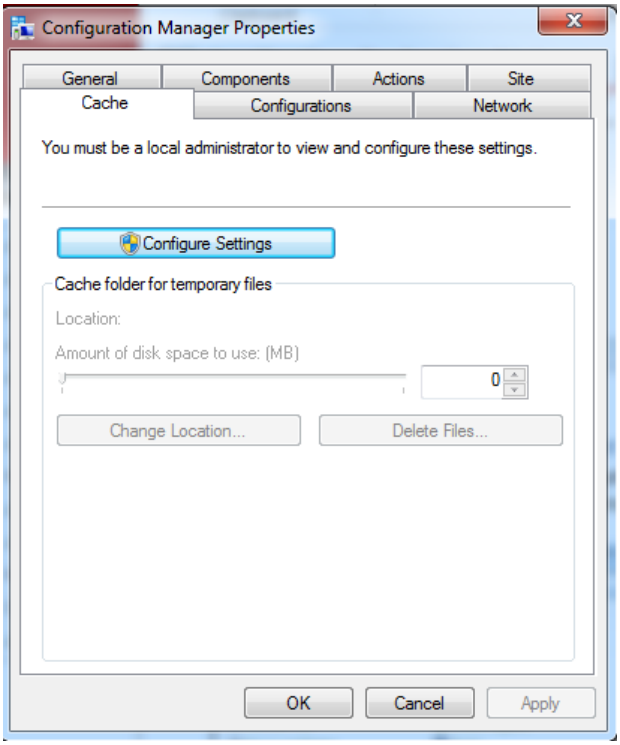
Clearing the Configuration Manager Cache

On the application packaging desktop test system:

- Open the **Configuration Manager** control panel applet.

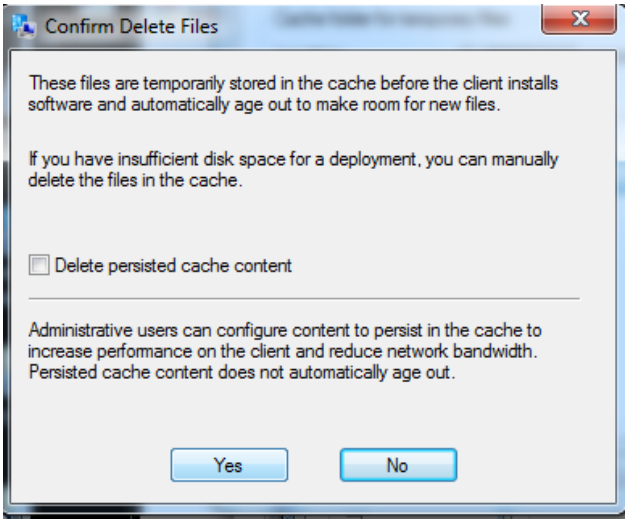


- Click the **Cache** tab in the *Configuration Manager Properties* window.



- Click the **Configure Settings** button and then the **Delete Files** button.

- Answer **Yes** on the *Confirm Delete Files* window.
    - **DO NOT** enable the *Delete persisted cache content* checkbox.



- Close the *Configuration Manager Properties* window.

Go to the top of this section and repeat the steps in the **6. Testing the Application Package** section.

Program Visibility Options

Issue: An application installs silently, but prompts the user for confirmation during the uninstall process.

The resulting application's SCCM package *User Experience* settings would be:

- Installation behavior: **Install for user**
- Logon requirement: **Whether or not a user is logged on**
- Installation program visibility: **Normal**

The program's visibility is set to **Normal** so that the uninstall confirmation pop-up message is displayed so the technician can acknowledge it and complete the uninstall process.

If visibility is set to **Hidden**, the uninstall confirmation pop-up message does not appear for the user to acknowledge and the uninstall never completes making it appear to hang.

If this happens, open **Task Manager** and end the uninstall process so that SCCM fails. Open the package properties, change the visibility to **Normal**, and update the content. Clear the target system's SCCM cache, and try uninstalling the application again. The uninstall confirmation pop-up message will appear for the technician to acknowledge and the uninstall will complete successfully.

PowerShell Scripting

PowerShell Issues

Some issues regarding scripting with PowerShell and SCCM:

- **Creating the Check**: Before any action is taken by a PowerShell script, a check must first be made to ensure that the action needs to be performed or skipped. For example, if the script 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 script full ownership of the system's files, folders, and registry for the time that the PowerShell script runs.

Note: The script 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 with out errors        #
# Privilege token only good for current PowerShell session                           #
#                                                                                    #
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2015  #
# 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 scripts 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 scripting 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 script to install the application silently.
- In the PowerShell install script:
  - 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 script with the **-ExecutionPolicy Bypass** switch:

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

Genericizing Scripts with Path Environment Variables

Windows system environment path variables are used in PowerShell install and uninstall scripts to lessen the amount of hard-coded paths in the scripts. 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:

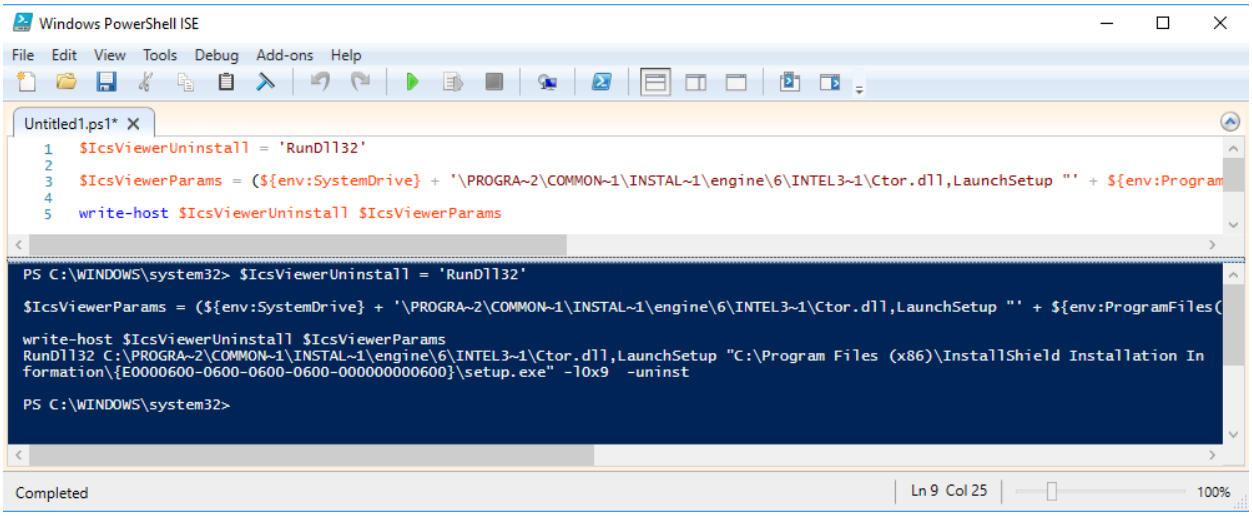
```
$AppName1Uninstall = '"'  + ${env:ProgramFiles} + '\AppName1\Uninstaller.EXE"'
```

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:

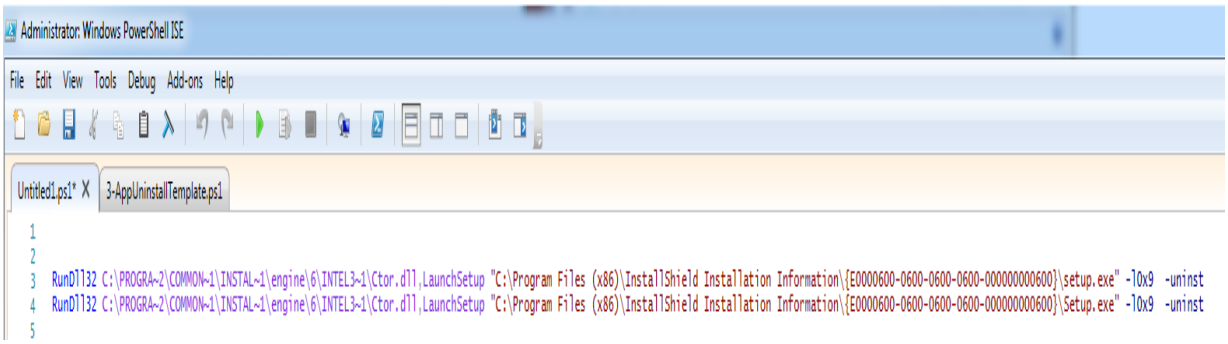
```
$IcsViewerUninstall = 'RunDll32'

$IcsViewerParams = (${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" -l0x9  -
     uninst')
```

Always watch out for placement of the double quotes around paths containing spaces. To check the output before running a script, copy the two string variables and past 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.



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 script pane and compare the two strings. Correct any errors and check the strings again.

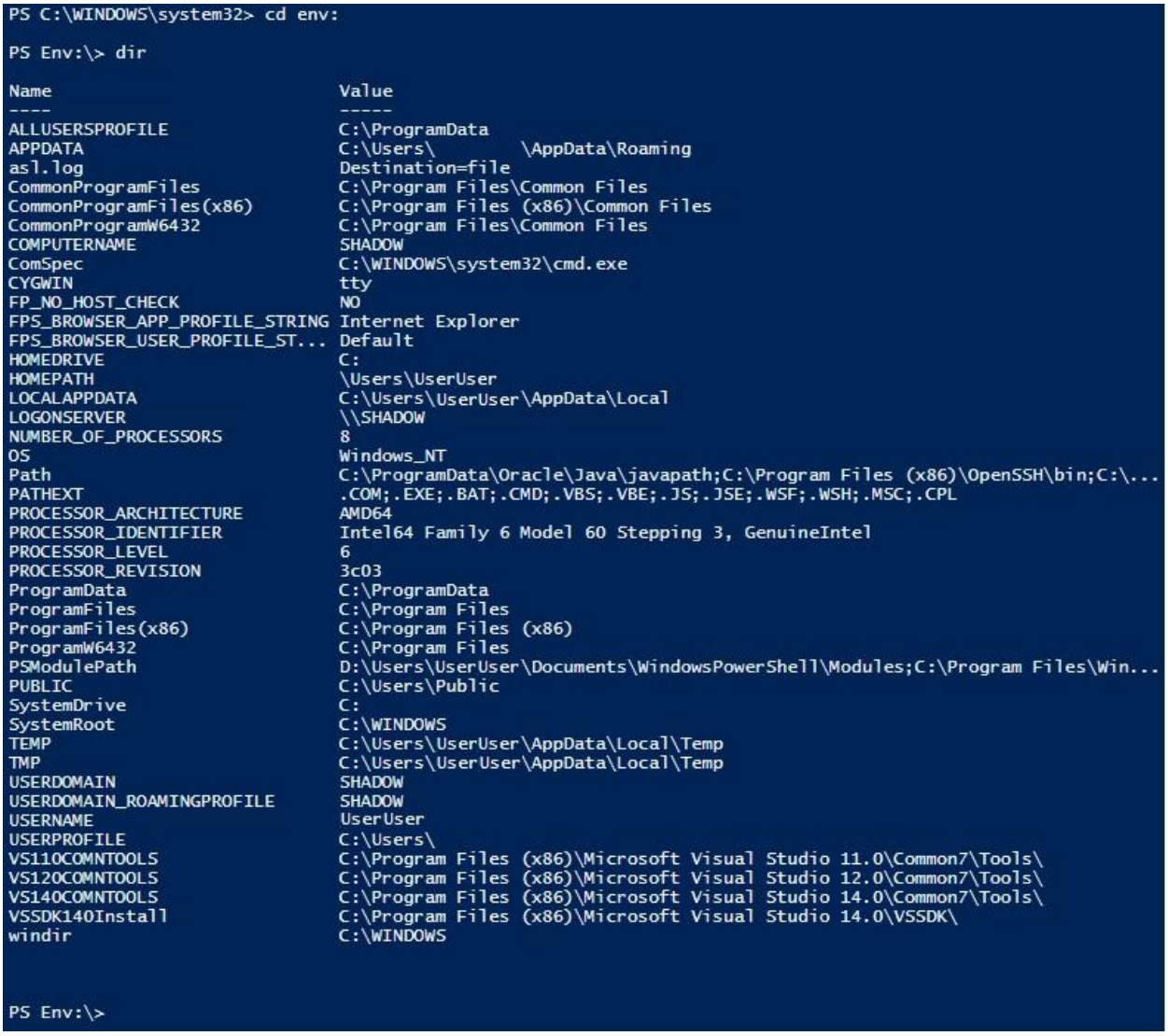


To find out what path environment variables are available, open a *PowerShell ISE* window with Administrator privileges (**Run ISE as Administrator**), switch to the ENV: drive, and list the directory.

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

Example screenshot for the most commonly used path environment variables in PowerShell:



```
PS C:\WINDOWS\system32> cd env:

PS Env:\> dir

Name                           Value
----                           -----
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\        \AppData\Roaming
asl.log                        Destination=file
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
CommonProgramW6432             C:\Program Files\Common Files
COMPUTERNAME                   SHADOW
ComSpec                        C:\WINDOWS\system32\cmd.exe
CYGWIN                         tty
FP_NO_HOST_CHECK               NO
FPS_BROWSER_APP_PROFILE_STRING Internet Explorer
FPS_BROWSER_USER_PROFILE_ST... Default
HOMEDRIVE                      C:
HOMEPATH                       \Users\UserUser
LOCALAPPDATA                   C:\Users\UserUser\AppData\Local
LOGONSERVER                    \\SHADOW
NUMBER_OF_PROCESSORS           8
OS                             Windows_NT
Path                           C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\OpenSSH\bin;C:\...
PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.CPL
PROCESSOR_ARCHITECTURE         AMD64
PROCESSOR_IDENTIFIER           Intel64 Family 6 Model 60 Stepping 3, GenuineIntel
PROCESSOR_LEVEL                6
PROCESSOR_REVISION             3c03
ProgramData                    C:\ProgramData
ProgramFiles                   C:\Program Files
ProgramFiles(x86)              C:\Program Files (x86)
ProgramW6432                   C:\Program Files
PSModulePath                   D:\Users\UserUser\Documents\WindowsPowerShell\Modules;C:\Program Files\Win...
PUBLIC                         C:\Users\Public
SystemDrive                    C:
SystemRoot                     C:\WINDOWS
TEMP                           C:\Users\UserUser\AppData\Local\Temp
TMP                            C:\Users\UserUser\AppData\Local\Temp
USERDOMAIN                     SHADOW
USERDOMAIN_ROAMINGPROFILE      SHADOW
USERNAME                       UserUser
USERPROFILE                    C:\Users\
VS110COMNTOOLS                 C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\Tools\
VS120COMNTOOLS                 C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\Tools\
VS140COMNTOOLS                 C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\Tools\
VSSDK140Install                C:\Program Files (x86)\Microsoft Visual Studio 14.0\VSSDK\
windir                         C:\WINDOWS


PS Env:\>
```

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 Method**s, **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 script 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 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`, *`AppName1`*`Uninstall.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 script:

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

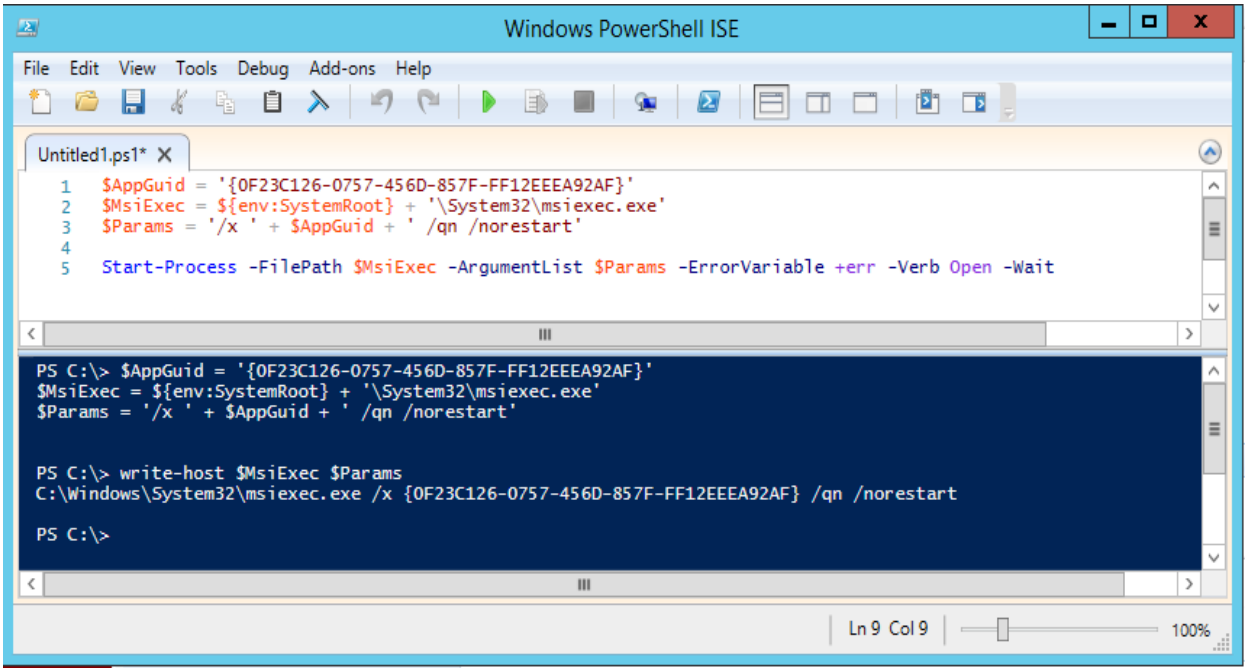
- **Verb**: Specifies a verb to use when starting the process. The verbs that are available are determined by the file name extension of the file that runs in the process. SCCM PowerShell scripts normally use the `-Verb Open` switch.

  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 script, 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.



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 Script

```
# 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 Script

```
# 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 Script

```
# 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 Script

```
# 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 Script

```
# 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 Script

```
# 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 Script

```
# 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 Script

```
# 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 Script

```
# 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

A 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 Script

```
# 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')
$errFileLocation =  (${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"
    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 $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
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}'

# 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}'

# 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}'

# 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')
$errFileLocation =  (${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 ""
    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 $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
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 regkey, 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 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** when needing to modify ACLs on system files and folders, including the commented credit to Precision Computing.

Add the *Adjusting Token Privileges* code, including the commented credit to Precision Computing, to enable the **SeDebugPrivilege** for installing SQL Server if blocked by domain Group Policy.

This code snippet works for both install and uninstall scripts. Place the snippet at the beginning of the install or uninstall script.

SeTakeOwnershipPrivilege

```
# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs with out errors        #
# Privilege token only good for current PowerShell session                           #
#                                                                                    #
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2015  #
# 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 with out errors   #
# Privilege token only good for current PowerShell session                      #

enable-privilege SeTakeOwnershipPrivilege |out-null
```

SeDebugPrivilege

```
# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs with out errors        #
# Privilege token only good for current PowerShell session                           #
#                                                                                    #
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2015  #
# 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 with out 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 -
       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
       -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 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 snipping 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 script 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 script 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
    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 ""

    $objAcl1 = Get-Acl $folder1
    $string = ('' + $objAcl1 + '.SetAccessRule(' + $objAr + ')')

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

    $objAcl1.SetAccessRule($objAr)

    Write-Host ""
    Write-Host "Setting new access list on $folder1 folder..."
    Write-Host ""
    Write-Host "Command: Set-Acl $folder1 $objAcl1"
    Write-Host ""

    Set-Acl $folder1 $objAcl1

    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 with out errors
# Privilege token only good for this PowerShell session
#
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2015  #
# 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-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 script 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')
$errFileLocation =  (${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 $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
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')
$errFileLocation =  (${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 $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
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')
$errFileLocation =  (${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 $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
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')
$errFileLocation =  (${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 $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
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
   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
       complete."
    Write-Host ""
}
Else
{
    Write-Host "    $DomainGroup domain group is already a 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
```

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
   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
            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 an PowerShell uninstall script 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 shortcuts.

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')
$errFileLocation =  (${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 $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""

timeout 5
```

Stopping Processes and Services

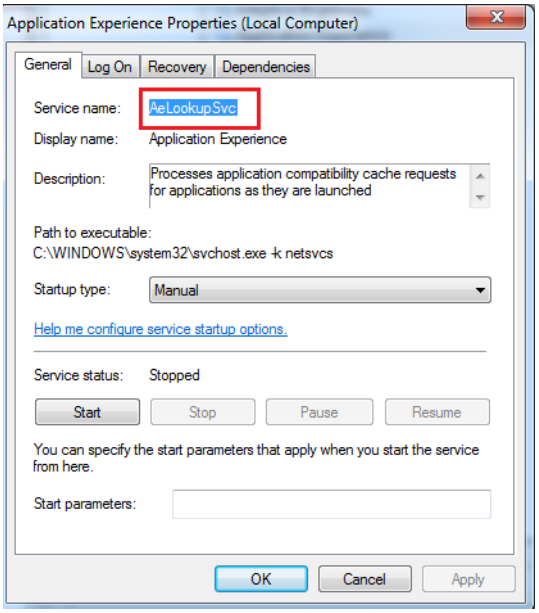
Uninstallers do not always stop an application's services and processes before uninstalling an application which always results in a failure.

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:

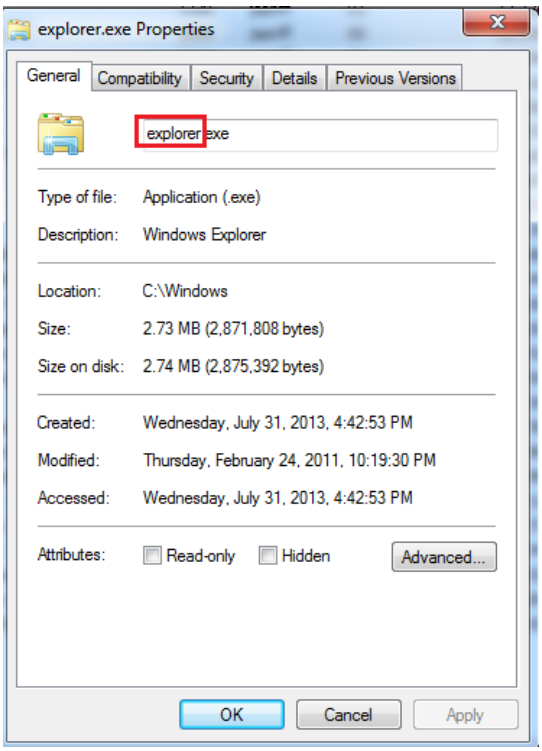
- Open the **Services** console.
- Double-click the service name in the listing to open the properties dialog.
- In the properties window, highlight the Service name and copy it into a PowerShell script variable.



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 script 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
   Mgr 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 has to do is start the application as all the required pieces are in place.

MSOW_PRD:

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

ODBC Data Sources:

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

Example SCCM Application Packages

Some actual packaged applications to demonstrate the above process.

7-Zip

Description
7-Zip is an open source, file archiver with a high compression ratio.

Package
Free software license.

Source file location: http://www.7-zip.org/download.html

Save latest MSI to: \\server\share\Sources\Software Vault\7zip

Import *7z920-x64.msi* into SCCM Applications as MSI Installer.

Install:
msiexec /i "7z920-x64.msi" /qn /norestart

Uninstall:
msiexec /x {23170F69-40C1-2702-0920-000001000000} /qn /norestart

Detection Method:
MSI Product Code: {23170F69-40C1-2702-0920-000001000000}

User Experience:
Behavior: Install for system
Logon: Whether or not a user is logged on
Visibility: Hidden
Enforce: No specific action

Notes

7-Zip
GUID: {23170F69-40C1-2702-0920-000001000000} (64-bit)

Registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{23170F69-40C1-2702-0920-000001000000}

UninstallString:
MsiExec.exe /I{23170F69-40C1-2702-0920-000001000000}

DisplayName: 7-Zip 9.20 (x64 edition)
DisplayVersion: 9.20.00.0
Publisher: Igor Pavlov

Alpha Five v6 - Manual

Description
Develop custom desktop and web applications quickly and easily with Alpha Five V6. With no programming skills required, Version 6's application scripting and web components allow you to take control of your information helping you build applications for your particular business or organisation. Alpha Five V6 works with a built-in DBF engine and MySQL, Oracle, SQL Server, DB2 and ODBC sources.

Category: Software Development.

Package
Single license package. License key required to activate product. For first time run, choose "Run as Administrator".

Source file location:
\\server\share\installs\Alpha 5 V6

Copy source files to:
\\server\share\Sources\Software Vault\Alpha 5 V6

Import *setup.exe* into SCCM Applications as a Script Installer.

Install:
"setup.exe"

Uninstall:
Powershell.exe –executionpolicy Bypass –file "AlphaFiveV6-Uninstall.ps1"

Detection Method:
Registry key exists.
Registry hive: HKLM
Registry key: SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Alpha Five V6

User Experience:
Behavior: Install for user
Logon: Only when a user is logged on
Visibility: Normal
Enforce: No specific action

Dependencies:
None

Notes
GUID: Alpha Five V6 (32-bit)

Registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Alpha Five V6

UninstallString:
C:\PROGRA~2\A5V6\UNWISE.EXE C:\PROGRA~2\A5V6\INSTALL.LOG

DisplayName: Alpha Five V6
DisplayVersion: 5.1 build 1532
Publisher: Alpha Software Inc.

AlphaFiveV6-Uninstall.ps1

```
#########################################################################
#                                                                       #
#                      Alpha Five v6 Uninstaller                        #
#                                                                       #
# Purpose: Remove the following components:                             #
#          - Uninstall Alpha Five v6                                    #
#                                                                       #
#########################################################################

# Declaration  of Variables.
$startLocation = Get-Location
$err=@()
$errorpath = '\AlphaFive_uninstall_error_log.txt'
$errFileLocation =  (${env:SystemDrive} + $errorpath)

#          - Uninstall Alpha Five v6                                    #
$AlphaFive = 'Alpha Five v6'
$AlphaFiveUninstall = ${env:SystemDrive} + '\PROGRA~2\A5V6\UNWISE.EXE'
$AlphaFiveParams = ${env:SystemDrive} + '\PROGRA~2\A5V6\INSTALL.LOG'
$AlphaFiveRegkey =
       'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Alpha
       Five V6'

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

#          - Uninstall Alpha Five v6                                  #
set-location $startLocation
Write-Host "*************************************************************"
Write-Host ""
Write-Host "Checking for $AlphaFive installation..."
Write-Host ""

If ((Test-Path -Path $AlphaFiveRegkey) -eq 'True')  # Check uninstall registry key or
       install path
{
    Write-Host "    $AlphaFive is installed."
    Write-Host ""
    Write-Host "Uninstalling $AlphaFive..."
    Write-Host ""
    Write-Host "Command:  $AlphaFiveUninstall $AlphaFiveParams"
    Write-Host ""
    Start-Process -FilePath $AlphaFiveUninstall -ArgumentList $AlphaFiveParams -
       ErrorVariable +err -Verb Open -Wait
    Write-Host ""
    Write-Host "    $AlphaFive uninstall complete."
    Write-Host ""
}
Else
```

```
{
    Write-Host "    $AlphaFive is not installed."
    Write-Host ""
}

sleep 5

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

timeout 5
```

Cyberlink PowerDVD - Manual

Description
A media player for Microsoft Windows providing DVD playback, with Blu-ray playback available in higher editions.

Package
Free software license.

Source file location:
\\server\share\installs\Cyberlink PowerDVD

Copy source files to:
\\server\share\Sources\Software Vault\Cyberlink PowerDVD

Import *Setup.exe* into SCCM Applications as a Script Installer.

Install:
"Setup.exe"

Uninstall:
Powershell.exe –executionpolicy Bypass –file "CyberlinkPowerDVD-Uninstall.ps1"

Detection Method:
Registry key exists.
Registry hive: HKLM
Registry key: SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{6811CAA0-BF12-11D4-9EA1-0050BAE317E1}

User Experience:
Behavior: Install for user
Logon: Only when a user is logged on
Visibility: Normal
Enforce: No specific action

Dependencies:
None

Notes
GUID: {6811CAA0-BF12-11D4-9EA1-0050BAE317E1} (32-bit)

Registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{6811CAA0-BF12-11D4-9EA1-0050BAE317E1}

UninstallString:

RunDll32
C:\PROGRA~2\COMMON~1\INSTAL~1\PROFES~1\RunTime\11\00\Intel32\Ctor.dll,LaunchSetup "C:\Program Files (x86)\InstallShield Installation Information\{6811CAA0-BF12-11D4-9EA1-0050BAE317E1}\Setup.exe" -l0x9 -cluninstall

DisplayName: PowerDVD DX
DisplayVersion: 8.3.5424
Publisher: CyberLink Corp.

CyberlinkPowerDVD-Uninstall.ps1

```
##########################################################################
#                                                                        #
#                 CyberLink PowerDVD DX Uninstaller                      #
#                                                                        #
# Purpose: Remove the following components:                              #
#          - Uninstall CyberLink PowerDVD DX                             #
#                                                                        #
##########################################################################

# Declaration  of Variables.
$startLocation = Get-Location
$err=@()
$errorpath = '\CyberLinkPowerDVD_uninstall_error_log.txt'
$errFileLocation =  (${env:SystemDrive} + $errorpath)

#          - Uninstall CyberLink PowerDVD DX                             #
$PowerDVD = 'CyberLink PowerDVD DX'
$PowerDVDUninstall = 'RunDll32'
$PowerDVDParams = (${env:SystemDrive} +
      '\PROGRA~2\COMMON~1\INSTAL~1\PROFES~1\RunTime\11\00\Intel32\Ctor.dll,LaunchSetu
      p "' + ${env:ProgramFiles(x86)} + '\InstallShield Installation
      Information\{6811CAA0-BF12-11D4-9EA1-0050BAE317E1}\Setup.exe" -l0x9  -
      cluninstall')
$PowerDVDRegkey =
      'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{6811CAA0
      -BF12-11D4-9EA1-0050BAE317E1}'

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

#          - Uninstall CyberLink PowerDVD DX                             #
set-location $startLocation
Write-Host "**************************************************************************"
Write-Host ""
Write-Host "Checking for $PowerDVD installation..."
Write-Host ""

If ((Test-Path -Path $PowerDVDRegkey) -eq 'True')  # Check uninstall registry key or
      install path
{
    Write-Host "    $PowerDVD is installed."
    Write-Host ""
    Write-Host "Uninstalling $PowerDVD..."
    Write-Host ""
    Write-Host "Command:  $PowerDVDUninstall $PowerDVDParams"
    Write-Host ""
    Start-Process -FilePath $PowerDVDUninstall -ArgumentList $PowerDVDParams -
      ErrorVariable +err -Verb Open -Wait
```

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

sleep 5

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

timeout 5
```

EasyLobby 10 - Manual

Description
HID Global's EasyLobby® Secure Visitor Management (SVM™) software is the main application for processing visitors, including ID scanning, record creation, badge printing, watch list screening, check-in and check-out, and email notification, among other features. The solution improves security by enabling organizations to identify exactly who is in their facilities, while enhancing the professionalism of an organization by streamlining the visitor check-in process and providing high quality visitor badges

Package
Single License package. License key required to complete setup. Scanner model required for ScanSnap driver and SDK installations.

Source file *EasyLobby100ISO.iso* location:
\\server\image$\EasyLobby

Use *Virtual CloneDrive* to open the ISO image and copy all of the files to the location below.

Extract ISO source files to:
\\server\share\Sources\Software Vault\EasyLobby10\Install

Create an *EasyLobby10-Install.ps1* PowerShell script and import into SCCM Applications as a Script Installer.

Create a second script, *EasyLobby10-Uninstall.ps1*, to uninstall all the EasyLobby 10 components.

Install:
Powershell.exe –executionpolicy Bypass –file "./EasyLobby10-Install.ps1"

Uninstall:
Powershell.exe –executionpolicy Bypass –file "./EasyLobby10-Uninstall.ps1"

Detection Method: Registry key exists.
Registry hive: HKLM
Registry key: SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\EasyLobby SVM
Value: DisplayVersion
Type: String
Equals: 10.0

User Experience:
Behavior: Install for user
Logon: Only when a user is logged on
Visibility: Normal
Enforce: No specific action

Dependencies:
None

Notes

EasyLobby SVM 10.0
Installs the following software:

- EasyLobby SVM
- SCCN SDK Version 9.50.19

Install string: EasyLobbySVM100Setup.exe

GUID: EasyLobby SVM

Registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\EasyLobby SVM

UninstallString:
%ProgramFiles(x86)%\EasyLobby\EasyLobby SVM 10.0\Uninstall.EXE

DisplayName: EasyLobby SVM
DisplayVersion: 10.0
Publisher: HID Global

ScanSnap Drivers 9.50
No entries appear in Programs and Features for any of the scanner drivers selected during setup.

Note: Uninstalls as part of EasyLobby SVM 10 uninstall which deletes the subfolder containing the ScanShell drivers.

Driver folders are located in:
C:\Program Files (x86)\EasyLobby\EasyLobby SVM 10.0\ScanShell

- SnapShell (default) installs two folders:
  - 64Bit
  - Backup
  - Snapshell Windows 7 64bit
- ScanShell 800 installs three folders:
  - Backup
  - ScanShell800 Windows 7 64bit
  - ScanShell800R Windows 7 64bit
- ScanShell 1000 installs one folder:
  - Backup
- ScanShell 1000A installs three folders:
  - Backup
  - ScanShell1000A Windows 7 64bit
  - ScanShell1000AN Windows 7 64bit

Install string: ScanShellDriversSetup950.exe

GUID: CSSN SDK Version 9.50

Registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\CSSN SDK Version 9.50.19

UninstallString:
C:\PROGRA~2\EASYLO~1\EASYLO~1.0\SCANSH~1\UNWISE.EXE
C:\PROGRA~2\EASYLO~1\EASYLO~1.0\SCANSH~1\INSTALL.LOG

- Note: Uninstalls as part of EasyLobby10-Uninstall

DisplayName: CSSN SDK
DisplayVersion: 9.50
Publisher: HID Global

CSSN SDK Version 9.50.19
Installs CSSN SDK Version 9.50.19 again, but with more options.

Install string: sdk_setup.exe

GUID: CSSN SDK Version 9.50.19

Registry key:
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\CSSN SDK Version 9.50.19

UninstallString: C:\PROGRA~2\CARDSC~1\SDK\UNWISE.EXE
C:\PROGRA~2\CARDSC~1\SDK\INSTALL.LOG

DisplayName: CSSN SDK Version 9.50.19
DisplayVersion: 9.50.19
Publisher: HID Global

EasyLobbySVM10-Install.ps1

```
################################################################################
#                                                                              #
#                  EasyLobby 10.0 PowerShell Installer Script                  #
#                                                                              #
#     Purpose: Installs the following components:                              #
#                        - EasyLobby SVM 10.0 and CSSN SDK 9.50.19             #
#                        - ScanShell Drivers 9.50                              #
#                        - CSSN SDK 9.50.19  with more options                 #
#                                                                              #
#               Script also does the following tasks:                          #
#                        - Copies  two files from a share to the local drive   #
#                        - Add Security domain group to local Power Users group#
#                        - Grant Power Users Full Control of three folders     #
#                        - Enable "Show hidden files, etc" in Folder Options   #
#                        - Copy three shortcuts to the Default User's Desktop  #
```

```
#                                                                                    #
######################################################################################

# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs with out errors        #
# Privilege token only good for current PowerShell session                           #
#                                                                                    #
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2015  #
# 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 ACLs with out errors   #
enable-privilege SeTakeOwnershipPrivilege |out-null

# Declaration  of Variables
$startLocation = Get-Location
$err=@()
$errorpath = '\EasyLobby10_install_error_log.txt'
$errFileLocation =  (${env:SystemDrive} + $errorpath)

#  Install EasyLobby SVM 10.0 and CSSN SDK 9.50.19               #
$EasyLobbySvmInstall = '.\EasyLobbySVM100Setup.exe'
$EasyLobbyRegkey =
       'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\EasyLobby
       SVM'

#  Install ScanShell Drivers 9.50                                 #
$ScanSnapDriversInstall = '.\ScanShellDriversSetup950.exe'
$ScanSnapInstallCheck = (${env:ProgramFiles(x86)} + '\EasyLobby\EasyLobby SVM
       10.0\ScanShell\Backup')

#  Install CSSN SDK Version 9.50.19 with more options             #
$CssnSdkInstall = '.\sdk_setup.exe'
$CssnSdkRegkey =
       'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\CSSN SDK
       9.50.19 Version 9.50.19'

#    - Copies two files from a share to the local drive           #
$CopyFile1name = '.\_options.el'
$CopyFile2name = '.\New logo.bmp'
$FileCopyDestination = (${env:ProgramFiles(x86)} + '\EasyLobby\EasyLobby SVM 10.0')

#    - Add Security Domain Group to the local Power Users group   #
$DomainGroup = 'Security'
$Domain = 'YOURDOMAIN'
$LocalGroup = 'Power Users'

#    - Grant Power Users Full Control of three folders            #
$folder1 = (${env:ProgramFiles(x86)} + '\EasyLobby')
$folder2 = (${env:windir} + '\Temp')
$folder3 = (${env:windir} + '\twain_32')

$User = 'BUILTIN\Power Users'
$FolderRights = 'FullControl'
$Flags = 'ContainerInherit,ObjectInherit'
$AllowDeny = 'Allow'

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

#    - Enable "Show hidden files, etc" in Folder Options          #
$ExplorerHiddenRegkey =
'HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced'

#    - Copy three shortcuts to the Default User's Desktop         #
$WshShell = New-Object -comObject WScript.Shell
$SourceExe1 = (${env:ProgramFiles(x86)} + '\EasyLobby\EasyLobby SVM
      10.0\Utilities\EmployeeImport.exe')
$SourceExe2 = (${env:ProgramFiles(x86)} + '\EasyLobby\EasyLobby SVM
      10.0\PhotoExport.exe')
$SourceExe3 = (${env:ProgramFiles(x86)} + '\EasyLobby\EasyLobby SVM
      10.0\EasyLobbySVM.EXE')
$DestinationPath1 = (${env:SystemDrive} + '\Users\Default\Desktop\EmployeeImport.lnk')
$DestinationPath2 = (${env:SystemDrive} + '\Users\Default\Desktop\PhotoExport.lnk')
$DestinationPath3 = (${env:SystemDrive} + '\Users\Default\Desktop\EasyLobbySVM.lnk')

Write-Host ""
Write-Host "************** EasyLobby 10.0 Installer ****************************"
Write-Host ""
Write-Host "Purpose: Installs the following components:"
Write-Host "         - EasyLobby SVM 10.0 and CSSN SDK 9.50.19 Version 9.50.19"
Write-Host "         - ScanShell Drivers 9.50"
Write-Host "         - CSSN SDK 9.50.19 Version 9.50.19 with more options"
Write-Host ""
Write-Host "Script also does the following tasks:"
Write-Host "         - Copies  two files from a share to the local drive"
Write-Host "         - Adds two users to the Power Users group"
Write-Host "         - Grants Power Users Full Control of three folders"
Write-Host "         - Enables 'Show hidden files, etc' in Folder Options"
Write-Host "         - Copies three shortcuts to the Default User's Desktop"
Write-Host ""

#  Install EasyLobby SVM 10.0 and CSSN SDK 9.50.19               #
set-location $startLocation
Write-Host "********************************************************************"
Write-Host ""
Write-Host "Checking for EasyLobby SVM 10.0 installation..."
Write-Host ""

If ((Test-Path -Path $EasyLobbyRegkey) -ne 'True')
{
    Write-Host "    EasyLobby SVM 10.0 is not installed."
    Write-Host ""
    Write-Host "Installing EasyLobby SVM 10.0..."
    Write-Host ""
    Write-Host "Command: " $EasyLobbySvmInstall
    Start-Process -FilePath $EasyLobbySvmInstall -ErrorVariable +err -Verb Open -Wait
    Write-Host ""
    Write-Host "    EasyLobby SVM 10.0 install complete."
    Write-Host ""
}
Else
{
    Write-Host "    EasyLobby SVM 10.0 already installed."
    Write-Host ""
}

Sleep 5

#  Install ScanShell Drivers 9.50                                #
set-location $startLocation
Write-Host "********************************************************************"
Write-Host ""
Write-Host "Checking for ScanShell Drivers 9.50 installation..."
Write-Host ""
```

```
If ((Test-Path -Path $ScanSnapInstallCheck) -ne 'True')
{
    Write-Host "    ScanShell Drivers 9.50 is not installed."
    Write-Host ""
    Write-Host "Installing ScanShell Drivers 9.50..."
    Write-Host ""
    Write-Host "Command: " $ScanSnapDriversInstall
    Start-Process -FilePath $ScanSnapDriversInstall -ErrorVariable +err -Verb Open -
       Wait
    Write-Host ""
    Write-Host "    ScanShell Drivers 9.50 install complete."
    Write-Host ""
}
Else
{
    Write-Host "    ScanShell Drivers 9.50 already installed."
    Write-Host ""
}

Sleep 5

#  Install CSSN SDK Version 9.50.19 with more options             #
set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for CSSN SDK 9.50.19 installation..."
Write-Host ""

If ((Test-Path -Path $CssnSdkRegkey) -ne 'True')
{
    Write-Host "    CSSN SDK 9.50.19 is not installed."
    Write-Host ""
    Write-Host "Installing CSSN SDK 9.50.19 ..."
    Write-Host ""
    Write-Host "Command: " $CssnSdkInstall
    Start-Process -FilePath $CssnSdkInstall -ErrorVariable +err -Verb Open -Wait
    Write-Host ""
    Write-Host "    CSSN SDK 9.50.19 install complete."
    Write-Host ""
}
Else
{
    Write-Host "    CSSN SDK 9.50.19 already installed."
    Write-Host ""
}

Sleep 5

#    - Copies two files from a share 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 $CopyFile1name $FileCopyDestination -Force"
Copy-Item $CopyFile1name $FileCopyDestination -Force
Write-Host ""
Write-Host "Command: Copy-Item $CopyFile2name $FileCopyDestination -Force"
Copy-Item $CopyFile2name $FileCopyDestination -Force
Write-Host ""
Write-Host "    File copy complete."
Write-Host ""

Sleep 5

#    - Add Security Domain Group to the local Power Users group   #
set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for Security group membership in local Power Users group..."
Write-Host ""
```

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

$string = '[ADSI]"WinNT://.' + $LocalGroup + ',group").Add("WinNT://' + $Domain + '/'
       + $DomainGroup + '")'

If ($r -match 'False')
{
    Write-Host "Adding Security Group to Power Users Group..."
    Write-Host ""
    Write-Host 'Command: ' $string
    Write-Host ""
    ([ADSI]"WinNT://./$LocalGroup,group").Add("WinNT://$Domain/$DomainGroup")
    Write-Host ""
    Write-Host "    Adding Security Group to Power Users complete."
    Write-Host ""
}
Else
{
    Write-Host "    Security group is already a member of local Power Users group"
    Write-Host ""
}

Sleep 5

#    - Grant Power Users Full Control of three folders            #
#    - Set ACL on EasyLobby folder                                #
set-location $startLocation
Write-Host "****************************************************************"
Write-Host ""
Write-Host "Checking ACL on $folder1..."
Write-Host ""

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

If ($r -eq $null)
{
    write-host "    $User group does not have $FolderRights permissions on $folder1"
    Write-Host ""

    $objAcl1 = Get-Acl $folder1
    $string = ('' + $objAcl1 + '.SetAccessRule(' + $objAr + ')')

    Write-Host ""
    Write-Host "Granting $User group $FolderRights for $folder1"
    Write-Host ""
    Write-Host "Command: " $string
    Write-Host ""
    $objAcl1.SetAccessRule($objAr)
    Write-Host ""
    Write-Host "Setting new access list on" $folder1
    Write-Host ""
    Write-Host "Command: Set-Acl" $folder1 $objAcl1
    Write-Host ""
    Set-Acl $folder1 $objAcl1
    Write-Host ""
    Write-Host "    New" $folder1 "access list set complete."
    Write-Host ""
}
Else
{
    Write-Host "    $User group already have $FolderRights for $folder1"
    Write-Host ""
}

Sleep 2

#    - Set ACL on Temp folder                                     #
set-location $startLocation
Write-Host "****************************************************************"
```

```
Write-Host ""
Write-Host "Checking ACL on $folder2..."
Write-Host ""

$r = (Get-Acl $folder2).Access | Where {$_.IdentityReference -eq $User -and
       $_.FileSystemRights -eq $FolderRights -and $_.AccessControlType -eq $AllowDeny}

If ($r -eq $null)
{
    write-host "    $User group does not have $FolderRights permissions on $folder2"
    Write-Host ""

    $objAcl2 = Get-Acl $folder2
    $string = ('' + $objAcl2 + '.SetAccessRule(' + $objAr + ')')

    Write-Host ""
    Write-Host "Granting $User group $FolderRights for $folder2"
    Write-Host ""
    Write-Host "Command: " $string
    Write-Host ""
    $objAcl2.SetAccessRule($objAr)
    Write-Host ""
    Write-Host "Setting new access list for" $folder2
    Write-Host ""
    Write-Host "Command: Set-Acl" $folder2 $objAcl2
    Write-Host ""
    Set-Acl $folder2 $objAcl2
    Write-Host ""
    Write-Host "    New $folder2 access list set complete."
    Write-Host ""
}
Else
{
    Write-Host "    $User group already have $FolderRights for $folder2"
    Write-Host ""
}

Sleep 2

#    - Set ACL on twain_32 folder                                    #
set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking ACL on $folder3..."
Write-Host ""

$r = (Get-Acl $folder3).Access | Where {$_.IdentityReference -eq $User -and
       $_.FileSystemRights -eq $FolderRights -and $_.AccessControlType -eq $AllowDeny}

If ($r -eq $null)
{
    write-host "    $User group does not have $FolderRights permissions on $folder3"
    Write-Host ""

    $objAcl3 = Get-Acl $folder3
    $string = ('' + $objAcl3 + '.SetAccessRule(' + $objAr + ')')

    Write-Host ""
    Write-Host "Granting $User group $FolderRights for $folder3"
    Write-Host ""
    Write-Host "Command: " $string
    Write-Host ""
    $objAcl3.SetAccessRule($objAr)
    Write-Host ""
    Write-Host "Setting new access list on" $folder3
    Write-Host ""
    Write-Host "Command: Set-Acl" $folder3 $objAcl3
    Write-Host ""
    Set-Acl $folder3 $objAcl3
    Write-Host ""
    Write-Host "    New" $folder3 "access list set complete."
    Write-Host ""
```

```
}
Else
{
    Write-Host "    $User group already have $FolderRights for $folder3"
    Write-Host ""
}

Sleep 5

#    - Enable "Show hidden files, etc" in Folder Options          #
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

#    - Copy three shortcuts to the Default User's Desktop         #
set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for 'EmployeeImport.exe' shortcut on Default User's Desktop..."
Write-Host ""

If ((Test-Path -Path $DestinationPath1) -ne 'True')
{
    Write-Host "    'EmployeeImport.exe' shortcut on Default User's Desktop does not
       exist."
    Write-Host ""
    Write-Host "Copying 'EmployeeImport.exe' shortcut Default User's Desktop..."
    Write-Host ""
    $Shortcut = $WshShell.CreateShortcut($DestinationPath1)
    $Shortcut.TargetPath = $SourceExe1
    $Shortcut.Save()
    Write-Host ""
    Write-Host "    Copying 'EmployeeImport.exe' shortcut complete."
    Write-Host ""
}
Else
{
    Write-Host "    'EmployeeImport.exe' shortcut already exists."
    Write-Host ""
}

Sleep 2

set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for 'PhotoExport.exe' shortcut on Default User's Desktop..."
Write-Host ""

If ((Test-Path -Path $DestinationPath2) -ne 'True')
{
    Write-Host "    'PhotoExport.exe' shortcut on Default User's Desktop does not
       exist."
```

```
    Write-Host ""
    Write-Host "Copying 'PhotoExport.exe' shortcut Default User's Desktop..."
    Write-Host ""
    $Shortcut = $WshShell.CreateShortcut($DestinationPath2)
    $Shortcut.TargetPath = $SourceExe2
    $Shortcut.Save()
    Write-Host ""
    Write-Host "    Copying 'PhotoExport.exe' shortcut complete."
    Write-Host ""
}
Else
{
    Write-Host "    'PhotoExport.exe' shortcut already exists."
    Write-Host ""
}

Sleep 2

set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for 'EasyLobbySVM.EXE' shortcut on Default User's Desktop..."
Write-Host ""

If ((Test-Path -Path $DestinationPath3) -ne 'True')
{
    Write-Host "    'EasyLobbySVM.EXE' shortcut on Default User's Desktop does not
       exist."
    Write-Host ""
    Write-Host "Copying 'EasyLobbySVM.EXE' shortcut Default User's Desktop..."
    Write-Host ""
    $Shortcut = $WshShell.CreateShortcut($DestinationPath3)
    $Shortcut.TargetPath = $SourceExe3
    $Shortcut.Save()
    Write-Host ""
    Write-Host "    Copying 'EasyLobbySVM.EXE' shortcut complete."
    Write-Host ""
}
Else
{
    Write-Host "    'EasyLobbySVM.EXE' shortcut already exists."
    Write-Host ""
}

Sleep 5

Write-Host "*********************************************************************"
Write-Host ""
Write-Host "    EasyLobby SVM 10.0 installation complete."
Write-Host ""
Write-Host "*********************************************************************"
Write-Host ""

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

Timeout 5
```

EasyLobbySVM10-Uninstall.ps1

```
##############################################################################
#                                                                            #
#                 EasyLobby 10.0 PowerShell Uninstall Script                 #
#                                                                            #
#     Purpose: Removes the following components:                             #
#                       - CSSN SDK Version 9.50.19                           #
#                       - ScanShell Drivers 9.50                             #
```

```
#                       - EasyLobby SVM 10.0                                    #
#                                                                               #
#              Script also undoes the following tasks:                          #
#                       - Delete leftover install folders                       #
#                       - Delete three shortcuts from the Default User's Desktop#
#                       - Remove Power Users Full Control of two folders        #
#                       - Remove Security group from local Power Users group    #
#                                                                               #
#################################################################################


# Enable 'SeTakeOwnershipPrivilege' to set system folder ACLs with out errors   #
# Privilege token only good for current PowerShell session                      #
#                                                                               #
# Adjusting Token Privileges with PowerShell by Precision Computing, copyright 2015  #
# 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 ACLs with out errors   #
enable-privilege SeTakeOwnershipPrivilege |out-null

#Declaration  of Variables
$startLocation = Get-Location
$err=@()
$errorpath = '\EasyLobby10_Uninstall_error_log.txt'
$errFileLocation =  (${env:SystemDrive} + $errorpath)

$CssnSdkRegkey =
       'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\CSSN SDK
       Version 9.50.19'
$CssnSdk2Uninstall = (${env:ProgramFiles(x86)} + '\Card Scanning
       Solutions\SDK\UNWISE.EXE')
$CssnSdk1Uninstall = (${env:ProgramFiles(x86)} + '\EasyLobby\EasyLobby SVM
       10.0\ScanShell\UNWISE.EXE')
$CardScanningFolder = (${env:ProgramFiles(x86)} + '\Card Scanning Solutions')

$EasyLobbyRegkey =
       'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\EasyLobby
       SVM'
$EasyLobbyUninstall = (${env:ProgramFiles(x86)} + '\EasyLobby\EasyLobby SVM
       10.0\Uninstall.EXE')
$EasyLobbyFolder = (${env:ProgramFiles(x86)} + '\EasyLobby')

$EmployeeImportShortcut = (${env:SystemDrive} +
       '\Users\Default\Desktop\EmployeeImport.lnk')
$PhotoExportShortcut  = (${env:SystemDrive} +
       '\Users\Default\Desktop\PhotoExport.lnk')
$EasyLobbyShortcut = (${env:SystemDrive} + '\Users\Default\Desktop\EasyLobbySVM.lnk')

$User = 'BUILTIN\Power Users'
$FolderRights = 'FullControl'
$Flags = 'ContainerInherit,ObjectInherit'
$AllowDeny = 'Allow'

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

$DomainGroup = 'Security'
$Domain = 'YOURDOMAIN'
$LocalGroup = 'Power Users'

$folder2 = (${env:windir} + '\Temp')
$folder3 = (${env:windir} + '\twain_32')

Write-Host ""
Write-Host "*************** EasyLobby 10.0 Uninstaller **************************"
Write-Host ""
Write-Host "Purpose: Removes the following components:"
Write-Host "         - CSSN SDK Version 9.50.19"
Write-Host "         - ScanShell Drivers 9.50"
Write-Host "         - EasyLobby SVM 10.0"
Write-Host ""
Write-Host "Script also undoes the following tasks:"
Write-Host "         - Delete leftover install folders"
Write-Host "         - Delete three shortcuts from the Default User's Desktop"
Write-Host "         - Remove Power Users Full Control of two folders"
Write-Host "         - Remove Security group from local Power Users group"
Write-Host ""

#  Uninstall CSSN SDKs Instance                                        #
set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for CSSN SDK installation..."
Write-Host ""

If ((Test-Path -Path $CssnSdkRegkey) -eq 'True')
{
    Write-Host "    CSSN SDK is installed."
    Write-Host ""
    $title =  "CSSN SDK Uninstall"
    $message = "Do you want to uninstall the CSSN SDK?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
       "Uninstall CSSN SDK"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
       CSSN SDK"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($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 CSSN SDK 2nd Instance..."
        Write-Host ""
        Write-Host "Command: " $CssnSdk2Uninstall
        Write-Host ""
        Start-Process -FilePath $CssnSdk2Uninstall -ErrorVariable +err -Verb Open -
             Wait
        Write-Host ""
        Write-Host "    CSSN SDK 2nd Instance uninstall complete."
        Write-Host ""

        Sleep 2

        Write-Host ""
        Write-Host "Uninstalling CSSN SDK 1st Instance..."
```

```
        Write-Host ""
        Write-Host "Command: " $CssnSdk1Uninstall
        Write-Host ""
        Start-Process -FilePath $CssnSdk1Uninstall -ErrorVariable +err -Verb Open -
             Wait
        Write-Host ""
        Write-Host "    CSSN SDK 1st Instance uninstall complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping CSSN SDK uninstall."
        Write-Host ""
    }
}
Else
{
    Write-Host "    CSSN SDK not installed."
    Write-Host ""
}

sleep 5

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

If ((Test-Path -Path $EasyLobbyRegkey) -eq 'True')
{
    Write-Host "    EasyLobby SVM is installed."
    Write-Host ""
    $title =  "EasyLobby SVM Uninstall"
    $message = "Do you want to uninstall EasyLobby SVM?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
       "Uninstall EasyLobby SVM"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
       EasyLobby SVM"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($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 EasyLobby SVM..."
        Write-Host ""
        Write-Host "Command: " $EasyLobbyUninstall
        Write-Host ""
        Start-Process -FilePath $EasyLobbyUninstall -ErrorVariable +err -Verb Open -
Wait
        Write-Host ""
        Write-Host "    EasyLobby SVM uninstall complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping EasyLobby SVM uninstall."
        Write-Host ""
    }
}
```

```
Else
{
    Write-Host "    EasyLobby SVM not installed."
    Write-Host ""
}

Sleep 5

# Delete Card Scanning Solutions folder in Program Files (x86)    #
set-location $startLocation
Write-Host "*****************************************************************"
Write-Host ""
Write-Host "Checking for Card Scanning Solutions install folder..."
Write-Host ""

if ((Test-Path -Path $CardScanningFolder) -eq 'True')
{
    Write-Host "    Card Scanning Solutions install folder exists."
    Write-Host ""
    $title =  "Card Scanning Solutions Install Folder"
    $message = "Do you want to delete the Card Scanning Solutions install folder?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
       "Delete Card Scanning Solutions install folder"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
       Card Scanning Solutions install folder"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($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 "Deleting Card Scanning Solutions install folder..."
        Write-Host ""
        Write-Host "Command:  Remove-Item -Path $CardScanningFolder -Recurse -Force"
        Write-Host ""
        Remove-Item -Path $CardScanningFolder -Recurse -Force
        Write-Host ""
        Write-Host "    Card Scanning Solutions install folder deletion complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping Card Scanning Solutions install folder deletion."
        Write-Host ""
    }
}
else
{
    Write-Host "    Card Scanning Solutions install folder does not exist."
    Write-Host ""
}

sleep 5

# Delete EasyLobby 10 install folder in Program Files (x86)       #
set-location $startLocation
Write-Host "*****************************************************************"
Write-Host ""
Write-Host "Checking for EasyLobby 10 install folder..."
Write-Host ""

if ((Test-Path -Path $EasyLobbyFolder) -eq 'True')
{
```

```
    Write-Host "    EasyLobby 10 install folder exists."
    Write-Host ""
    $title =  "EasyLobby 10 Install Folder"
    $message = "Do you want to delete the EasyLobby 10 install folder?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
       "Delete EasyLobby 10 install folder"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
       EasyLobby 10 install folder"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($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 "Deleting EasyLobby 10 install folder..."
        Write-Host ""
        Write-Host "Command:  Remove-Item -Path $EasyLobbyFolder -Recurse -Force"
        Write-Host ""
        Remove-Item -Path $EasyLobbyFolder -Recurse -Force
        Write-Host ""
        Write-Host "    EasyLobby 10 install folder deletion complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping EasyLobby 10 install folder deletion."
        Write-Host ""
    }
}
else
{
    Write-Host "    EasyLobby 10 install folder does not exist."
    Write-Host ""
}

sleep 5

#    - Delete three shortcuts from the Default User's Desktop     #
set-location $startLocation
Write-Host "*****************************************************************"
Write-Host ""
Write-Host "Checking for Employee Import shortcut on Default Desktop..."
Write-Host ""

if ((Test-Path -Path $EmployeeImportShortcut) -eq 'True')
{
    Write-Host ""
    Write-Host "Removing Employee Import shortcut.."
    Write-Host ""
    Write-Host "Command:  Remove-Item -Path $EmployeeImportShortcut -Force"
    Write-Host ""
    Remove-Item -Path $EmployeeImportShortcut -Force
    Write-Host ""
    Write-Host "    Employee Import shortcut removal complete."
    Write-Host ""
}
else
{
    Write-Host "    Employee Import shortcut does not exist."
    Write-Host ""
}

sleep 2
```

```
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for Photo Export shortcut on Default Desktop..."
Write-Host ""

if ((Test-Path -Path $PhotoExportShortcut) -eq 'True')
{
    Write-Host ""
    Write-Host "Removing Photo Export shortcut.."
    Write-Host ""
    Write-Host "Command:  Remove-Item -Path $PhotoExportShortcut -Force"
    Write-Host ""
    Remove-Item -Path $PhotoExportShortcut -Force
    Write-Host ""
    Write-Host "    Photo Export shortcut removal complete."
    Write-Host ""
}
else
{
    Write-Host "    Photo Export shortcut does not exist."
    Write-Host ""
}

sleep 2

Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for Easy Lobby shortcut on Default Desktop..."
Write-Host ""

if ((Test-Path -Path $EasyLobbyShortcut) -eq 'True')
{
    Write-Host ""
    Write-Host "Removing Easy Lobby shortcut.."
    Write-Host ""
    Write-Host "Command:  Remove-Item -Path $EasyLobbyShortcut -Force"
    Write-Host ""
    Remove-Item -Path $EasyLobbyShortcut -Force
    Write-Host ""
    Write-Host "    Easy Lobby shortcut removal complete."
    Write-Host ""
}
else
{
    Write-Host "    Easy Lobby shortcut does not exist."
    Write-Host ""
}

sleep 5

#     - Check Power User access on temp folder                    #
set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking ACL on $folder2..."
Write-Host ""

$r = (Get-Acl $folder2).Access | Where {$_.IdentityReference -eq $User -and
      $_.FileSystemRights -eq $FolderRights -and $_.AccessControlType -eq $AllowDeny}

If ($r -ne $null)
{
    Write-Host "    $User group group has $FolderRights permissions for $folder2"
    Write-Host ""
    $title =  "Remove $User Permissions"
    $message = "Do you want to remove the $User from the $folder2 folder permissions?"
    $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 $folder2
        $string = ('' + $objAcl2 + '.RemoveAccessRule(' + $objAr + ')')
        Write-Host ""
        Write-Host "Removing $User group from $folder2 folder access list..."
        Write-Host ""
        Write-Host 'Command: ' $string
        Write-Host ""
        $objAcl2.RemoveAccessRule($objAr)
        Write-Host ""
        Write-Host "Setting new access list on $folder2 folder..."
        Write-Host ""
        Write-Host "Command: Set-Acl $folder2 $objAcl2"
        Write-Host ""
        Set-Acl $folder2 $objAcl2
        Write-Host ""
        Write-Host "    New $folder2 folder access list set complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping ACL check on $folder2 folder."
        Write-Host ""
    }
}
Else
{
    Write-Host "    $User group does not have $FolderRights permissions for $folder2
       folder."
    Write-Host ""
}

sleep 5

#    - Check Power User access on twain_32 folder                    #
set-location $startLocation
Write-Host "****************************************************************"
Write-Host ""
Write-Host "Checking ACL on $folder3..."
Write-Host ""

$r = (Get-Acl $folder3).Access | Where {$_.IdentityReference -eq $User -and
       $_.FileSystemRights -eq $FolderRights -and $_.AccessControlType -eq $AllowDeny}

If ($r -ne $null)
{
    Write-Host "    $User group group has $FolderRights permissions for $folder3"
    Write-Host ""
    $title =  "Remove $User Permissions"
    $message = "Do you want to remove the $User from the folder $folder3 permissions?"
    $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)
    {
        $objAcl3 = Get-Acl $folder3
        $string = ('' + $objAcl3 + '.RemoveAccessRule(' + $objAr + ')')
        Write-Host ""
        Write-Host "Removing $User group from $folder3 folder access list..."
        Write-Host ""
        Write-Host 'Command: ' $string
        Write-Host ""
        $objAcl3.RemoveAccessRule($objAr)
        Write-Host ""
        Write-Host "Setting new access list on $folder3 folder..."
        Write-Host ""
        Write-Host "Command: Set-Acl $folder3 $objAcl3"
        Write-Host ""
        Set-Acl $folder3 $objAcl3
        Write-Host ""
        Write-Host "    New $folder3 folder access list set complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping ACL check on $folder3 folder."
        Write-Host ""
    }
}
Else
{
    Write-Host "    $User group does not have $FolderRights permissions for $folder3
       folder."
    Write-Host ""
}

sleep 5

#    - Remove Security domain group from the Power Users group                #
set-location $startLocation
Write-Host "*********************************************************************"
Write-Host ""
Write-Host "Checking for Security group membership in local Power Users group..."
Write-Host ""

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

$string = '[ADSI]"WinNT://.' + $LocalGroup + ',group").Remove("WinNT://' + $Domain +
      '/' + $DomainGroup + '")'

If ($r -match 'True')
{
    Write-Host "    Security group is member of local Power Users grup."
    Write-Host ""
    $title =  "Security Group Removal"
    $message = "Do you want to remove the Security group from the local Power Users
       group?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
       "Remove Security group"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
       Security 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)
    {
        Write-Host ""
        Write-Host "Removing Security group from local Power Users group..."
        Write-Host ""
        Write-Host 'Command: ' $string
        Write-Host ""
        ([ADSI]"WinNT://./$LocalGroup,group").Remove("WinNT://$Domain/$DomainGroup")
        Write-Host ""
        Write-Host "    Security group removal complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping Security group removal."
        Write-Host ""
    }
}
Else
{
    Write-Host "    Security group is not member of local Power Users group."
    Write-Host ""
}

sleep 5

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

timeout 5
```

MSOW - Manual

Description
MSO for the Web (MSOW) is a comprehensive, web-based credentialing and priviledging system by Morresey Associates. MSOW combines the speed and flexibility of the Internet with Morrisey's advanced technology to automate physician and allied health professional credentialing.

Package
Enterprise license. **Office 2010 Professional must be uninstalled first!** Install Office 2003 Word 2003, then Office 2003 Access 2003 SP2. Script copies files to local hard drive folder, creates shortcut on All Users desktop, and creates the MSOW_PRD database 32-bit ODBC connection.

Source file location:
\\serverdb01\MSOW

Copy source files to:
\\server\share\Sources\Software Vault\MSOW

Import *MSOW-Install.ps1* into SCCM Applications as a Script Installer.

Install:
Powershell.exe –executionpolicy Bypass –file "MSOW-Install.ps1"

Uninstall:
Powershell.exe –executionpolicy Bypass –file "MSOW-Uninstall.ps1"

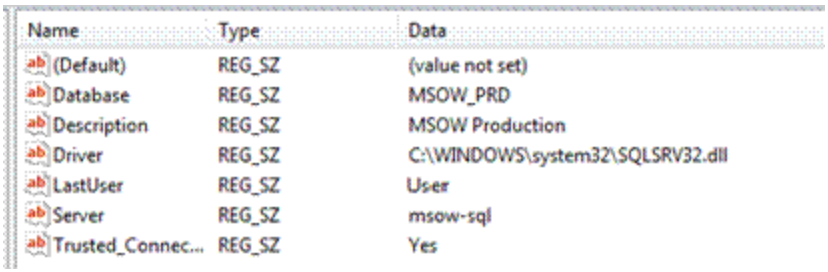
Detection Method:
File exists.
Path: %SystemDrive%\Morrisey\MSOW
Target: MSOWPrivs_PRD.mdb

User Experience:
Behavior: Install for user
Logon: Only when a user is logged on
Visibility: Normal
Enforce: No specific action

Dependencies:
None

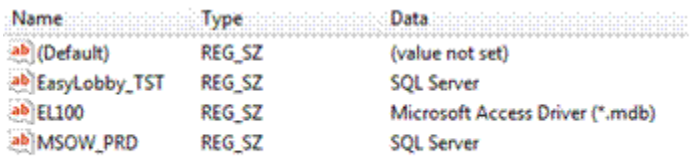
Setup Notes

MSOW_PRD Regkey:



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

ODBC Data Sources Regkey:



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

MSOW-Install.ps1

```
#############################################################################
#                                                                           #
#                            MSOW Setup Script                              #
#                                                                           #
# Purpose: Install the following components:                                #
#          - Verify Word 2003 is installed                                  #
#          - Verify Access 2003 is installed                                #
#          - Create folder off of local hard drive root                     #
#          - Copy two files to the local drive                              #
#          - Copy shortcut to ALL USERS desktop                             #
#          - Create a 32-bit ODBC connection                                #
#                                                                           #
#############################################################################

# Declaration  of Variables
$startLocation = Get-Location
$err=@()
$errorpath = '\MSOW_install_error_log.txt'
$errFileLocation =  (${env:SystemDrive} + $errorpath)

$MSOW = 'MSOW'

#          - Verify Word 2003 is installed                                  #
$Word2003 = 'Microsoft Office Word 2003'
$Word2003Regkey =
      'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{901B0409
      -6000-11D3-8CFE-0150048383C9}'

#          - Verify Access 2003 is installed                                #
$Access2003 = 'Microsoft Office Access 2003 SP2'
$Access2003Regkey =
      'HKLM:SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{90150409
      -6000-11D3-8CFE-0150048383C9}'

#          - Create folder off of local hard drive root                     #
$RootFolder = ${env:SystemDrive} + '\Morrisey'
$InstallFolder = ${env:SystemDrive} + '\Morrisey\MSOW'

#          - Copy two files to the local drive                              #
$CopyFileName1 = '.\MSOWfoo_PRD.mdb'
$CopyFileName2 = '.\Foo01.ICO'
```

```
$FileCopyDestination = (${env:SystemDrive} + '\Morrisey\MSOW\')

#          - Copy shortcut to ALL USERS desktop                              #
$SourcePath1 = ('.\MSOW foo.lnk')
$DestinationPath1 = (${env:SystemDrive} + '\Users\Public\Desktop\MSOW foo.lnk')

#          - Created a 32-bit ODBC connection                                #
$ConnectionName = 'MSOW_PRD'
$ConnectionDescription = 'MSOW PRODUCTION'
$SqlServer = 'MSOW-SQL'
$SqlDatabase = 'MSOW_PRD'
$SqlDriver = '"C:\WINDOWS\System32\sqlsrv32.dll"'
$HKLMPath1 = “HKLM:SOFTWARE\Wow6432Node\ODBC\ODBC.INI\” + $ConnectionName
$HKLMPath2 = “HKLM:SOFTWARE\Wow6432Node\ODBC\ODBC.INI\ODBC Data Sources”

Write-Host ""
Write-Host "*************************************************************"
Write-Host ""
Write-Host "$MSOW Setup Script"
Write-Host ""
Write-Host "Purpose: Performs the following tasks:"
Write-Host "          - Check for $Word2003 installation"
Write-Host "          - Check for $Access2003 installation"
Write-Host "          - Copy two files to the local drive"
Write-Host "          - Copy shortcut to ALL USERS desktop"
Write-Host "          - Create a $ConnectionName 32-bit ODBC connection"
Write-Host ""

#          - Verify Word 2003 is installed                                   #
set-location $startLocation
Write-Host "*************************************************************"
Write-Host ""
Write-Host "Checking for $Word2003 installation..."
Write-Host ""

If ((Test-Path -Path $Word2003Regkey) -ne 'True')  # Check uninstall registry key or
       install path
{
    Write-Host "    $Word2003 is not installed!"
    Write-Host ""
    Write-Host "Please install $Word2003 before continuing..."
    Write-Host ""
    Write-Host "    Press any key to quit..."
    Pause
    Break
}
Else
{
    Write-Host "    $Word2003 already installed."
    Write-Host ""

    # Get Word 2003 InstallLocation from registry
    $Word2003Location = (Get-ItemProperty -Path $Word2003Regkey -Name
       InstallLocation).InstallLocation

    Write-Host "$Word2003 located at:  $Word2003Location"
    Write-Host ""
}

Sleep 5

#          - Verify Access 2003 is installed                                 #
set-location $startLocation
Write-Host "*************************************************************"
Write-Host ""
Write-Host "Checking for $Access2003 installation..."
Write-Host ""

If ((Test-Path -Path $Access2003Regkey) -ne 'True')  # Check uninstall registry key or
       install path
{
    Write-Host "    $Access2003 is not installed!"
```

```
    Write-Host ""
    Write-Host "Please install $Access2003 before continuing..."
    Write-Host ""
    Write-Host "    Press any key to quit..."
    Pause
    Break
}
Else
{
    Write-Host "    $Access2003 already installed."
    Write-Host ""

    # Get Word 2003 InstallLocation from registry
    $Access2003Location = (Get-ItemProperty -Path $Access2003Regkey -Name
       InstallLocation).InstallLocation

    Write-Host "$Access2003 located at:  $Access2003Location"
    Write-Host ""
}

Sleep 5

#          - Create folder off of local hard drive root                  #
set-location $startLocation
Write-Host "**********************************************************************"
Write-Host ""
Write-Host "Checking for $RootFolder on local drive..."
Write-Host ""

If ((Test-Path -Path $RootFolder) -ne 'True')  # Check uninstall registry key or
       install path
{
    Write-Host "    $RootFolder does not exist."
    Write-Host ""
    Write-Host "Creating $RootFolder on local drive..."
    Write-Host ""
    Write-Host "Command: md $RootFolder"
    md $RootFolder
    Write-Host ""
    Write-Host "    $RootFolder creation complete."
    Write-Host ""
}
Else
{
    Write-Host "    $RootFolder already exists."
    Write-Host ""
}

Sleep 2

#          - Create folder off of local hard drive root                  #
set-location $startLocation
Write-Host "**********************************************************************"
Write-Host ""
Write-Host "Checking for $InstallFolder on local drive..."
Write-Host ""

If ((Test-Path -Path $InstallFolder) -ne 'True')  # Check uninstall registry key or
       install path
{
    Write-Host "    $InstallFolder does not exist."
    Write-Host ""
    Write-Host "Creating $InstallFolder on local drive..."
    Write-Host ""
    Write-Host "Command: md $InstallFolder"
    md $InstallFolder
    Write-Host ""
    Write-Host "    $InstallFolder creation complete."
    Write-Host ""
}
Else
{
```

```
    Write-Host "    $InstallFolder already exists."
    Write-Host ""
}

Sleep 2

#          - Copy two files 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 2

#          - Copy shortcut to ALL USERS desktop                         #
set-location $startLocation
Write-Host "**********************************************************************"
Write-Host ""
Write-Host "Copying shortcut to ALL USERS desktop..."
Write-Host ""
Write-Host "Command: Copy-Item $SourcePath1 $DestinationPath1 -Force"
Copy-Item $SourcePath1 $DestinationPath1 -Force
Write-Host ""
Write-Host "    Shortcut copy complete."
Write-Host ""

Sleep 2

#          - Created 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

    ## This is required to allow the ODBC connection to show up in the ODBC
       Administrator application.
    md $HKLMPath2 -ErrorAction silentlycontinue

    set-itemproperty -path $HKLMPath2 -name “$ConnectionName” -value 'SQL Server'

    Write-Host ""
    Write-Host "    Creating $ConnectionName 32-bit ODBC connection complete."
    Write-Host ""
    Write-Host "To confirm 32-bit ODBC connection creation, run odbcad32.exe from the
       C:\Windows\SysWOW64 folder"
```

```
    Write-Host ""
}
Else
{
    Write-Host "    $ConnectionName 32-bit ODBC connection already exists."
    Write-Host ""
}

Sleep 5

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

Timeout 5
```

MSOW-Uninstall.ps1

```
#########################################################################
#                                                                       #
#                          MSOW Removal Script                          #
#                                                                       #
# Purpose: Install the following components:                            #
#          - Delete install folder off of local hard drive root         #
#          - Delete shortcut on ALL USERS desktop                       #
#          - Delete the 32-bit ODBC connection                          #
#                                                                       #
#########################################################################

# Declaration  of Variables
$startLocation = Get-Location
$err=@()
$errorpath = '\MSOW_uninstall_error_log.txt'
$errFileLocation =  (${env:SystemDrive} + $errorpath)
$MSOW = 'MSOW'

#          - Delete install folder off of local hard drive root         #
$InstallFolder = ${env:SystemDrive} + '\Morrisey\MSOW'

#          - Delete shortcut on ALL USERS desktop                       #
$Shortcut = 'MSOW foo'
$DestinationPath1 = (${env:SystemDrive} + '\Users\Public\Desktop\MSOW foo.lnk')

#          - Delete the 32-bit ODBC connection                          #
$ConnectionName = 'MSOW_PRD'
$ConnectionDescription = 'MSOW PRODUCTION'
$SqlServer = 'MSOW-SQL'
$SqlDatabase = 'MSOW_PRD'
$SqlDriver = '"C:\WINDOWS\System32\sqlsrv32.dll"'
$HKLMPath1 = “HKLM:SOFTWARE\Wow6432Node\ODBC\ODBC.INI\” + $ConnectionName
$HKLMPath2 = “HKLM:SOFTWARE\Wow6432Node\ODBC\ODBC.INI\ODBC Data Sources”

Write-Host ""
Write-Host "*************************************************************"
Write-Host ""
Write-Host "$MSOW Uninstaller"
Write-Host ""
Write-Host "Purpose: Performs the following tasks:"
Write-Host "         - Delete $MSOW install folder off of local hard drive root"
Write-Host "         - Delete $Shortcut shortcut on ALL USERS desktop"
Write-Host "         - Delete $ConnectionName 32-bit ODBC connection"
Write-Host ""

#          - Delete install folder off of local hard drive root         #
set-location $startLocation
Write-Host "*************************************************************"
Write-Host ""
Write-Host "Checking for $MSOW install folder..."
```

```
Write-Host ""

if ((Test-Path -Path $InstallFolder) -eq 'True')
{
    Write-Host "    $MSOW install folder exists."
    Write-Host ""
    $title =  "$MSOW Install Folder"
    $message = "Do you want to delete the $MSOW install folder?"
    $yes = New-Object System.Management.Automation.Host.ChoiceDescription "&Yes",
       "Delete $MSOW install folder"
    $no = New-Object System.Management.Automation.Host.ChoiceDescription "&No", "Leave
       $MSOW install folder"
    $options = [System.Management.Automation.Host.ChoiceDescription[]]($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 "Deleting $MSOW install folder..."
        Write-Host ""
        Write-Host "Command:  Remove-Item -Path $InstallFolder -Recurse -Force"
        Write-Host ""
        Remove-Item -Path $InstallFolder -Recurse -Force
        Write-Host ""
        Write-Host "    $MSOW install folder deletion complete."
        Write-Host ""
    }
    Else
    {
        Write-Host ""
        Write-Host "    Skipping $MSOW install folder deletion."
        Write-Host ""
    }
}
else
{
    Write-Host "    $MSOW install folder does not exist."
    Write-Host ""
}

sleep 5

#          - Delete shortcut on ALL USERS desktop                          #
set-location $startLocation
Write-Host "*************************************************************************"
Write-Host ""
Write-Host "Checking for $Shortcut shortcut on ALL USERS desktop..."
Write-Host ""

if ((Test-Path -Path $DestinationPath1) -eq 'True')
{
    Write-Host ""
    Write-Host "Removing $Shortcut shortcut.."
    Write-Host ""
    Write-Host "Command:  Remove-Item -Path $DestinationPath1 -Force"
    Write-Host ""
    Remove-Item -Path $DestinationPath1 -Force
    Write-Host ""
    Write-Host "    $Shortcut shortcut removal complete."
    Write-Host ""
}
else
{
    Write-Host "    $Shortcut shortcut does not exist."
```

```
    Write-Host ""
}

sleep 5

#          - 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

        ## This is required to remove the ODBC connection from showing up in the ODBC
             Administrator application.
        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

#    - Indicate location of error log file                                  #
$err | Out-File $errFileLocation
Write-Host "*** Log file location = $errFileLocation ***"
Write-Host ""
Timeout 5
```

Surface Pro 3 - November 2014 Driver Package

Description
November 2014 driver release for Surface Pro 3.

Package
Free software license.

Source file location:
\\childrens\files\Sources\Software Vault\Microsoft Surface\OOBNovember18th2014SurfacePro3

Import *InstallDrivers.ps1* into SCCM Applications as a Script Installer.

Install:
Powershell.exe –executionpolicy Bypass –file "InstallDrivers.ps1"

Uninstall:
N/A

Detection Method: PowerShell Script

```
$driverlist = @{"Surface Pro System Aggregator Firmware" = "3.9.350.0";
                "Surface Pro Embedded Controller Firmware" = "38.7.50.0";
                "Microsoft LifeCam Front" = "5.20.1034.0";
                "Surface Ethernet Adapter" = "8.14.0704.2014";
                "Surface Pro UEFI" = "3.11.350.0";
                "Surface Pro Touch Controller Firmware" = "426.27.66.0";
                "Intel(R) Serial IO I2C Host Controller - 9C61" = "1.1.165.1";
                "Intel(R) Serial IO I2C Host Controller - 9C62" = "1.1.165.1";
                "Intel(R) Serial IO GPIO Host Controller" = "1.1.165.1";
                "Intel(R) Display Audio"= "6.16.0.3135";
                "Intel(R) Management Engine Interface*" = "9.5.24.1790";
                "Intel(R) HD Graphics Family" = "10.18.10.3496";
                "Intel(R) 8 Series SATA AHCI Controller - 9C03" = "9.4.0.1023";
                 "Intel(R) 8 Series LPC Controller (Premium SKU) - 9C43" =
                    "9.4.0.1023";
                "Intel(R) 8 Series PCI Express Root Port #3 - 9C14" = "9.4.0.1023";
                "Intel(R) 8 Series SMBus Controller - 9C22" = "9.4.0.1023";
                "Marvell AVASTAR Wireless-AC Network Controller" = "15.68.3066.135";
                "Marvell AVASTAR Bluetooth Radio Adapter" = "15.68.3066.135";
                "Realtek High Definition Audio" = "6.0.1.7198";
                "Realtek USB 3.0 Card Reader" = "6.2.9200.30164";
                "Surface Accessory Device" = "2.0.1012.0";
                "Surface Cover Audio" = "2.0.722.0";
                "Surface Cover Click" = "2.0.375.0";
                "Surface Type Cover" = "2.0.364.0";
                "Surface Touch Cover" = "2.0.722.0";
                "Surface Type Cover Fw Update" = "2.0.722.0";
                "Surface Touch Cover FW Update" = "2.0.722.0";
                "Surface Type Cover 2 Fw Update" = "2.0.722.0";
                "Surface Touch Cover2 FW Update" = "2.0.722.0";
                "Surface Type Cover 3 Firmware Update" = "2.0.1021.0";
                "Surface Display Calibration" = "2.0.1002.0";
                "Surface Intergration" = "2.0.1168.0";
                "Surface Home Button" = "2.0.1174.0";
                "Surface Cover Telemetry" = "2.0.722.0";
                "Surface Pen Driver" = "2.5.14.0";
                "Surface Pen" = "1.0.13.0" }

$LogPath = ${env:SystemRoot} + '\Logs\'
$File = 'SurfaceProDriversNeeded.txt'
```

```
$Flag = 0

foreach ($key in $driverlist.GetEnumerator() | Sort-Object Name)
{
    If (Get-WmiObject win32_pnpsigneddriver | where {$_.DeviceName -eq $key.Name})
    {
        If (!(Get-WmiObject win32_pnpsigneddriver | where {$_.DeviceName -eq $key.Name
             -and $_.DriverVersion -eq $key.Value}))
        {
            $m = $key.Name  + ' is not current:  ' + $key.Value
            Add-Content -Path $LogPath$File $m
            $Flag = $Flag + 1
        }
    }
}

If ($Flag -gt 0)
{
    $false
}
Else
{
    if (Test-Path -path $LogPath$File)
    {
        rm $LogPath$File -force
    }
    $true
}
```

NOTE: The driver search key name for the `"Intel(R) Management Engine Interface*"` driver requires a wildcard character at the end of the name due to the driver name in the `.INF` file containing a blank space at the end. Without the wildcard character to account for the blank space in the name, the driver check will fail every time. Every driver's `.INF` file should be cracked open to compare against the driver key name string for accuracy.

User Experience:
Behavior: Install for system
Logon: Whether or not a user is logged on
Visibility: Hidden
Enforce: No specific action

Requirements:
Category: Device
Condition: Operating system
One of: All Windows 8.1

Dependencies:
None

InstallDrivers.ps1

```
$startLocation = Get-Location
$Scriptpath = Split-Path -parent $startLocation
$files = get-childitem -path $Scriptpath -recurse -filter *.inf

foreach ($file in $files)
{
```

```
    $PnpUtilCmd = 'pnputil.exe'

    $PnpUtilParams = '-i -a "' + $file.FullName + '"'

    Start-Process -FilePath $PnpUtilCmd -ArgumentList $PnpUtilParams -ErrorVariable
       +err -Verb Open -Wait
}
```