PSAppDeployToolkit

Administrator Guide

v3.8.4 - 26th January 2021

PowerShell App Deployment Toolkit Website

Introduction

Toolkit Overview

The PowerShell App Deployment Toolkit provides a set of functions to perform common application deployment tasks and to interact with the user during a deployment. It simplifies the complex scripting challenges of deploying applications in the enterprise, provides a consistent deployment experience and improves installation success rates.

The PowerShell App Deployment Toolkit can be used to replace your WiseScript, VBScript and Batch script wrappers with one versatile, re-usable and extensible tool.

Features

- **Easy To Use** - Any PowerShell beginner can use the template and the functions provided with the Toolkit to perform application deployments.
- **Consistent** - Provides a consistent look and feel for all application deployments, regardless of complexity.
- **Powerful** - Provides a set of functions to perform common deployment tasks, such as installing or removing multiple applications, prompting users to close applications, setting registry keys, copying files, etc.
- **User Interface** - Provides user interaction through , customizable user interface dialog boxes, progress dialogs and balloon tip notifications that can all be branded with custom logo and banner.
- **Localized** - The UI is localized in several languages and more can easily be added using the XML configuration file.
- **Integration** - Integrates well with Microsoft MEMCM / SCCM / Configuration Manager, providing installation and uninstall deployment types with options on how to handle exit codes, such as suppressing reboots or returning a fast retry code.
- **Update-able** - The logic engine and functions are separated from per-application scripts, so that you can update the toolkit when a new version is released and maintain backwards compatibility with your deployment scripts.
- **Extensible** - The Toolkit can be easily extended to add custom scripts and functions.

- **Helpful** - The Toolkit provides detailed logging of all actions performed and even includes a graphical console to browse the help documentation for the Toolkit functions.

System Requirements and Support

The v3 release of the PowerShell App Deployment Toolkit has been developed (and tested) to work with a wide range of Operating Systems from Windows XP to Windows 10 (and their Windows Server equivalents) in order to provide enterprise-wide compatibility.

The system requirements are as follows:

- PowerShell 2.0
- Windows NT 5.1 and above

While we have attempted to maintain this backwards compatibility through the life cycle of the toolkit, the degree of testing performed across older Operating Systems such as XP and Vista is limited as the bulk of testing is performed on the latest OS versions. However, the toolkit has widespread adoption in the enterprise from SMEs to large multinationals so there is safety in numbers and the assurance that the toolkit has been put through its paces VERY extensively on a large number of Windows clients around the globe.

We have no way of knowing exactly how many Windows clients it has been used on, but sometime in 2016 we were aware of the PowerShell App Deployment Toolkit being used to deploy software on a rough count of 3.5 million endpoints (based on companies we know using it and their total number of staff, or where we have explicitly given that number for a company).

Donations

We have invested our spare time in the creation and ongoing development, maintenance and support of this community tool on a voluntary basis - it is not part of our day jobs. Donations to the project are welcome, please visit the following page for details on making a contribution:

http://psappdeploytoolkit.com/donate

Licensing

PowerShell App Deployment Toolkit - Provides a set of functions to perform common application deployment tasks on Windows.

Copyright (C) 2020 - Sean Lillis / Dan Cunningham.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this program. If not, see http://www.gnu.org/licenses/.

Toolkit Functionality

User Experience Features

- An interface to prompt the user to close specified applications that are open prior to starting the application deployment. The user is prompted to save their documents and has the option to close the programs themselves, have the toolkit close the programs, or optionally defer. Optionally, a countdown can be displayed until the applications are automatically closed.
- The ability to allow the user to defer an installation X number of times, X number of days or until a deadline date is reached.
- The ability to prevent the user from launching the applications that need to be closed while the application installation is in progress.
- An indeterminate progress dialog with customizable message text that can be updated throughout the deployment.
- A restart prompt with an option to restart later or restart now and a countdown to automatic restart.
- The ability to notify the user if disk space requirements are not met.
- Custom dialog boxes with options to customize title, text, buttons & icon.
- Balloon tip notifications to indicate the beginning and end of an installation and the success or failure of an installation.
- Branding of the above UI components using a custom logo icon and banner for your own Organization.
- The ability to run in interactive, silent (no dialogs) or non-interactive mode (default for running MEMCM task sequence or session 0).
- The UI is localized into several languages and more can be easily added using the XML configuration file.

Custom PowerShell Cmdlets to simplify deployments

- Provides extensive logging of both the Toolkit functions and any MSI installation / uninstallation.
- Provides the ability to execute any type of setup (MSI or EXEs) and handle the return codes.
- Mass remove MSI applications with a partial match (e.g. remove all versions of all MSI applications which match "Office")
- Perform MEMCM actions such as Machine and User Policy Refresh, Inventory Update and Software Update
- Supports installation of applications on Citrix / Remote Desktop Session Host Servers
- Update Group Policy
- Copy / Delete Files
- Get / Set / Remove Registry Keys and Values
- Get / Set INI File Keys and Values
- Check File versions
- Pin or Unpin applications to the Start Menu or Task Bar
- Create Start Menu Shortcuts
- Register / Unregister DLL files
- Refresh desktop icons / environment variables
- Test network connectivity
- Test power connectivity
- Check whether a PowerPoint slide show is running in full screen presentation mode

MEMCM / SCCM / Config Manager Integration

- Handles MEMCM exit codes, including time sensitive dialogs supporting "MEMCM Fast Retry" - providing more accurate Reporting (no more Failed due to timeout errors).
- Ability to prevent reboot codes (3010) from being passed back to MEMCM, which would cause a reboot prompt.
- Supports the MEMCM application model by providing an install and uninstall deployment type for every deployment script.
- Bundle multiple application installations to overcome the supported limit of 5 applications in the MEMCM application dependency chain.
- Compared to compiled deployment packages, e.g. WiseScript, the Toolkit utilizes the MEMCM cache correctly and MEMCM Distribution Point bandwidth more efficiently by using loose files.

Help

- A graphical console to easily browse the Toolkit Cmdlet documentation.
- Further documentation available on GitHub Wiki.
- Complete documentation included as PDF for use on e-readers / offline.

Toolkit Composition

Files Structure

The toolkit is comprised of the following files:

File Name	Description
Deploy-Application.ps1	Performs the actual install / uninstall and is the only file that needs to be modified, depending on your level of customization.
Deploy-*Application.exe*	An optional executable that can be used to launch the *Deploy-Application.ps1* script without opening a PowerShell console window. Supports passing command-line parameters to the script.
AppDeployToolkitMain.ps1	Contains all of the functions and logic used by the installation script. By Separating the logic from the installation script, we can obfuscate away the complex code and make enhancements independently of the installation scripts that contain per-application actions.
AppDeployToolkitConfig.xml	Contains configurable options referenced by the *AppDeployToolkitMain.ps1* script, such as MSI switches and User Interface messages, which are customizable and localized in several languages. This is intended to be a static file that is configured once, not on a per-application basis.
AppDeployToolkitExtensions.ps1	This is an optional PowerShell script that can be used to extend the toolkit functionality with custom functions. It is automatically dot-sourced by the *AppDeployToolkitMain.ps1* script.
AppDeployToolkitHelp.ps1	This is a script that displays a help console to browse the functions included in the Toolkit and copy and paste examples in to your deployment script.

Folders Structure

The Root folder contains the *Deploy-Application.exe* and *Deploy-Application.ps1* files. The *Deploy-Application.ps1* file is the only file that should be modified on a per-application basis.

The directories below contain the installation files and supporting files referenced by the toolkit.

Folder Name	Description
AppDeployToolkit	Contains the Toolkit dependency files.
Files	Contains the primary installation files, e.g. MSI file.
SupportFiles	Contains any supporting resources or assets, e.g. files you need to copy to the target machine using the Toolkit during deployment.

Toolkit User Interface

The user interface consists of several components detailed below. The user interface can be branded with a custom logo and banner.

All of the UI components include message text that is customizable in the *AppDeployToolkitConfig.xml*. The UI has been localized in 11 different languages: English, French, Spanish, Portuguese, German, Italian, Dutch, Swedish, Danish, Norwegian and Japanese. Additional languages can be easily added in the XML configuration file.

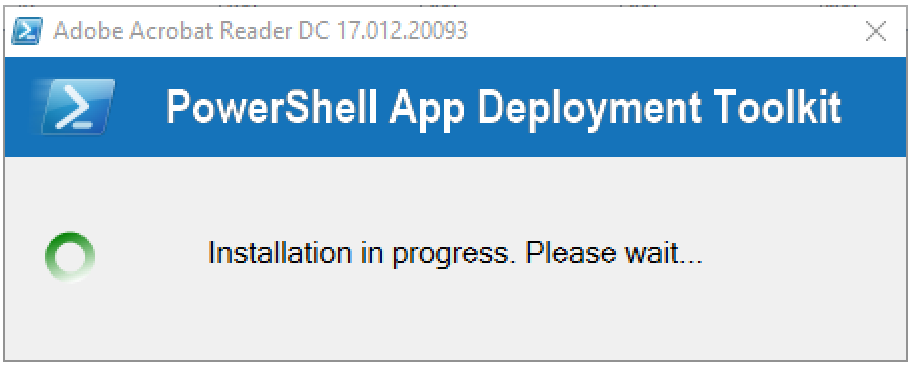
The language used by the Toolkit UI is selected automatically based on the language culture of the operating system, so the same *AppDeployToolkitConfig.xml* file can be used in a multi-language environment.

The user interface can be suppressed by specifying the deploy mode parameter as follows:

```
Deploy-Application.ps1 -DeployMode "Silent"
```

Installation Progress

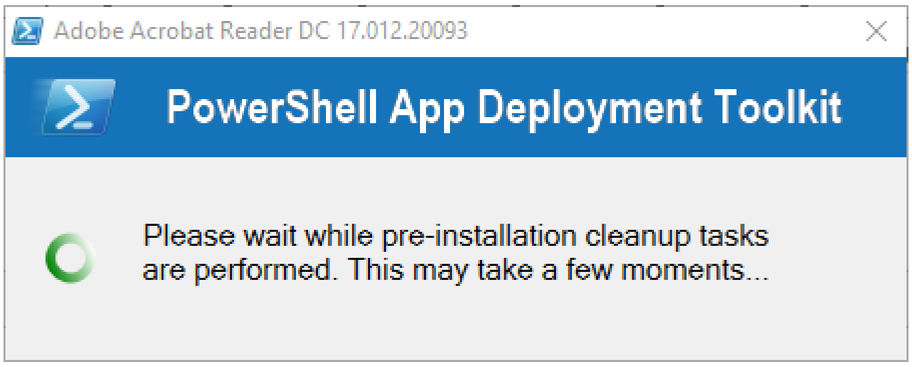
The installation progress message displays an indeterminate progress ring to indicate an installation is in progress and display status messages to the end user. This is invoked using the `Show-InstallationProgress` Function.


Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

Installation in progress. Please wait...

The progress message can be dynamically updated to indicate the stage of the installation or to display custom messages to the user, using the `Show-InstallationProgress` function.

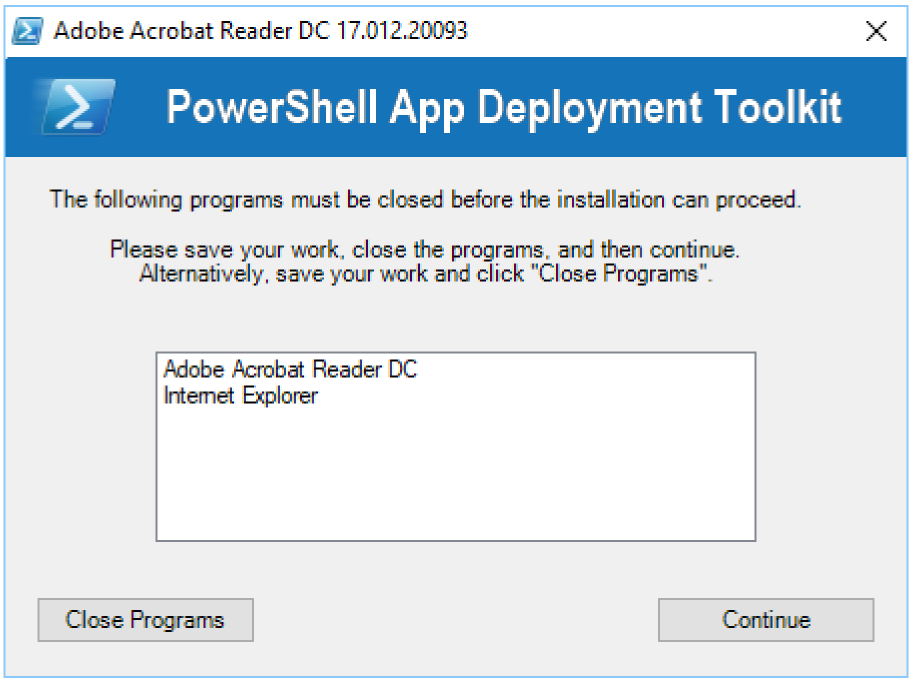

Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

Please wait while pre-installation cleanup tasks are performed. This may take a few moments...

Installation Welcome Prompt

The application welcome prompt can be used to display applications that need to be closed, an option to defer and a countdown to closing applications automatically. Use the `Show-InstallationWelcome` function to display the prompts shown below.


Adobe Acrobat Reader DC 17.012.20093

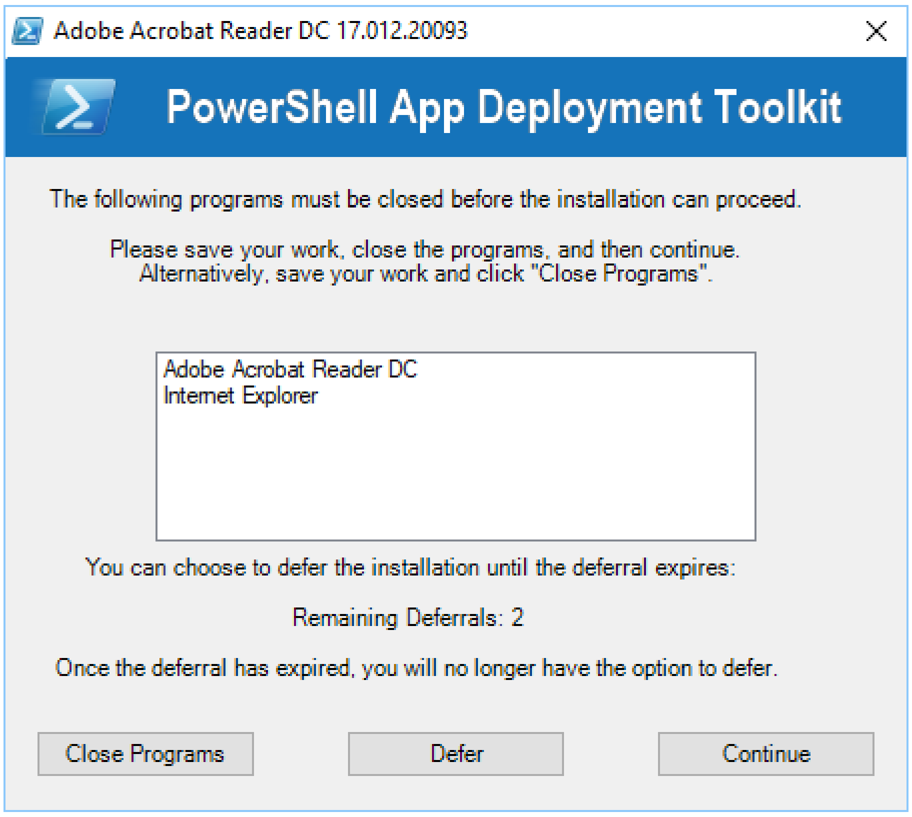
PowerShell App Deployment Toolkit

The following programs must be closed before the installation can proceed.

Please save your work, close the programs, and then continue.
Alternatively, save your work and click "Close Programs".

Adobe Acrobat Reader DC
Internet Explorer

Welcome prompt with close programs option and defer option:


Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

The following programs must be closed before the installation can proceed.

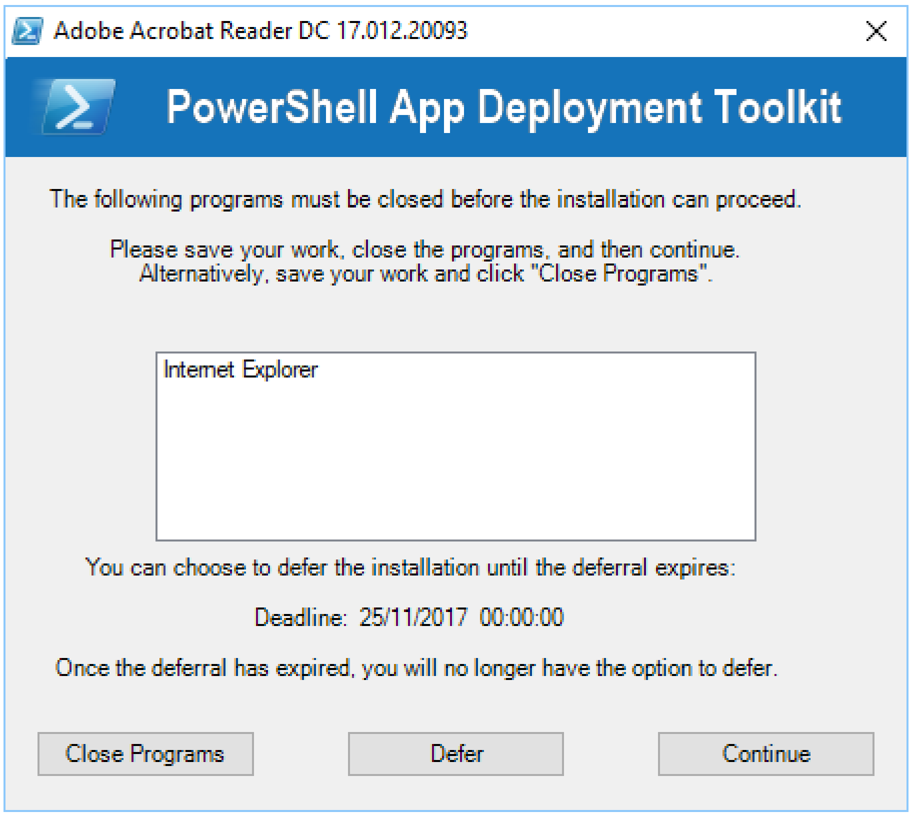
Please save your work, close the programs, and then continue.
Alternatively, save your work and click "Close Programs".

Adobe Acrobat Reader DC
Internet Explorer

You can choose to defer the installation until the deferral expires:

Remaining Deferrals: 2

Once the deferral has expired, you will no longer have the option to defer.



Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

The following programs must be closed before the installation can proceed.

Please save your work, close the programs, and then continue.
Alternatively, save your work and click "Close Programs".

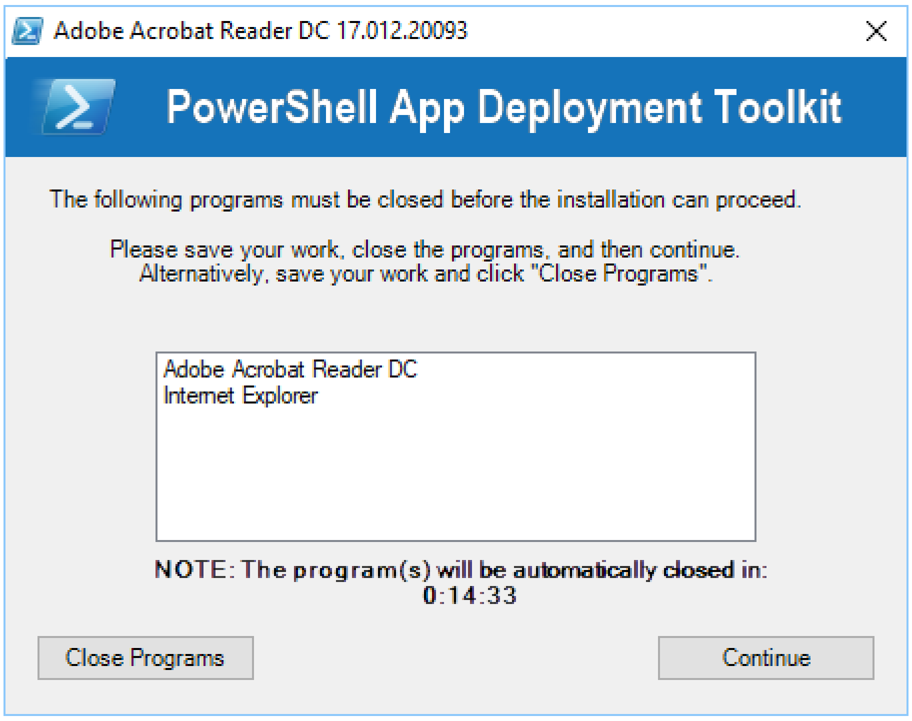
Internet Explorer

You can choose to defer the installation until the deferral expires:

Deadline: 25/11/2017 00:00:00

Once the deferral has expired, you will no longer have the option to defer.

Welcome prompt with close programs options and countdown to automatic closing of applications:



Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

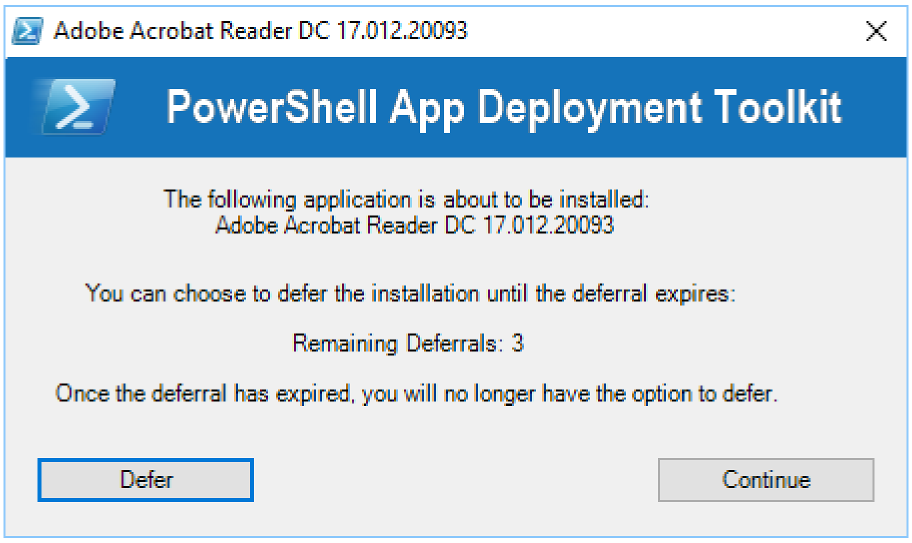
The following programs must be closed before the installation can proceed.

Please save your work, close the programs, and then continue.
Alternatively, save your work and click "Close Programs".

Adobe Acrobat Reader DC
Internet Explorer

NOTE: The program(s) will be automatically closed in:
0:14:33

Welcome prompt with just a defer option:


Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

The following application is about to be installed:
Adobe Acrobat Reader DC 17.012.20093

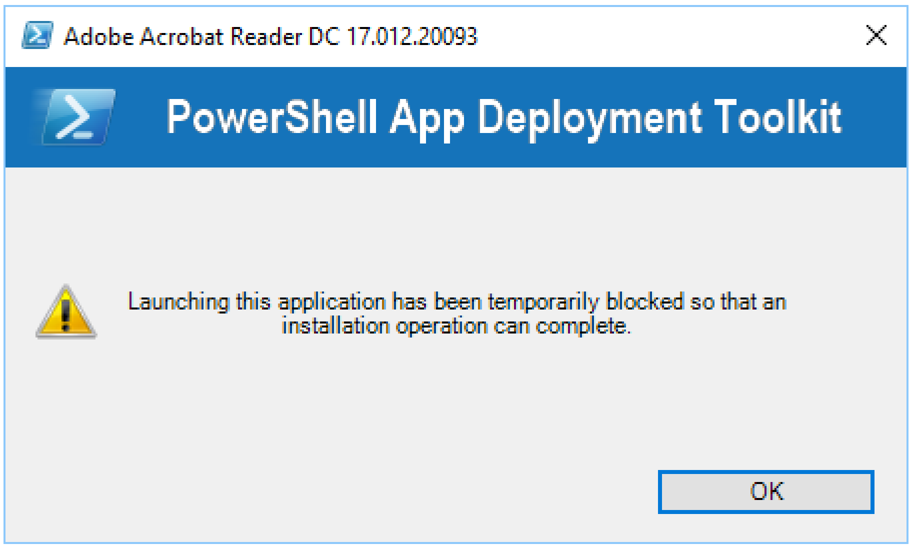
You can choose to defer the installation until the deferral expires:

Remaining Deferrals: 3

Once the deferral has expired, you will no longer have the option to defer.

Block Application Execution

If the block execution option is enabled (see `Show-InstallationWelcome` function), the user will be prompted that they cannot launch the specified application(s) while the installation is in progress. The application will be unblocked again once the installation has completed.

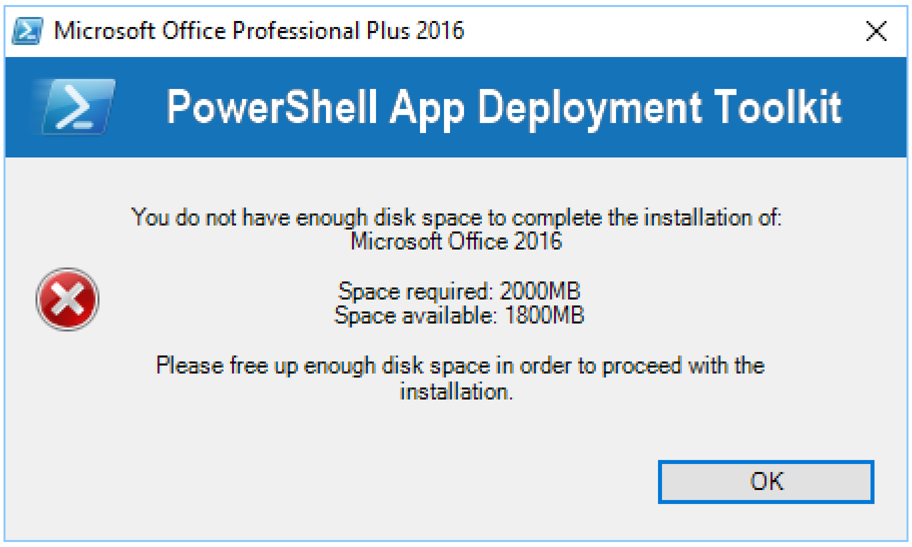

Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

Launching this application has been temporarily blocked so that an installation operation can complete.

Disk Space Requirements

If the `CheckDiskSpace` parameter is used with the `Show-InstallationWelcome` function and the disk space requirements are not met, the following prompt will be displayed and the installation will not proceed.


Microsoft Office Professional Plus 2016

PowerShell App Deployment Toolkit

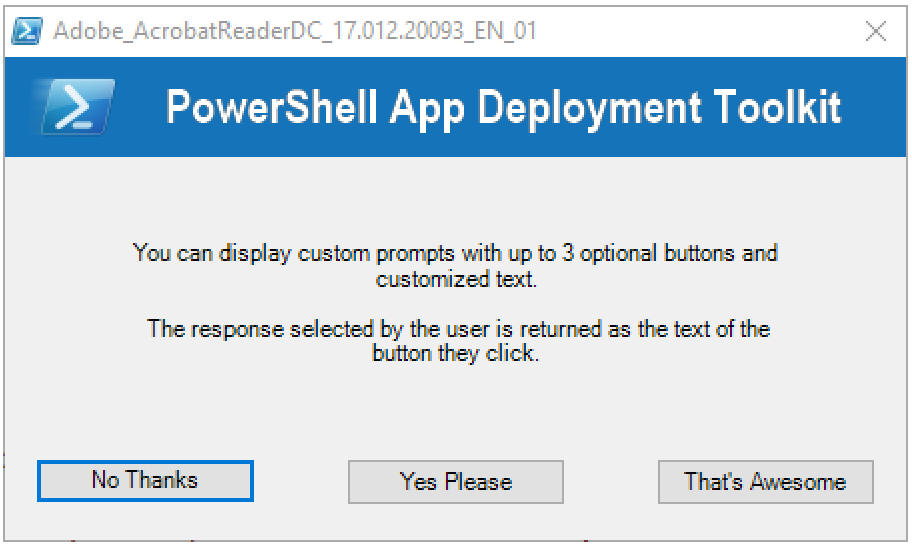
You do not have enough disk space to complete the installation of:
Microsoft Office 2016

Space required: 2000MB
Space available: 1800MB

Please free up enough disk space in order to proceed with the installation.

Custom Installation Prompt

A custom prompt with the toolkit branding can be used to display messages and interact with the user using the `Show-InstallationPrompt` function. The title and text is customizable and up to 3 customizable buttons can be included on the prompt as well as optional system icons, e.g.

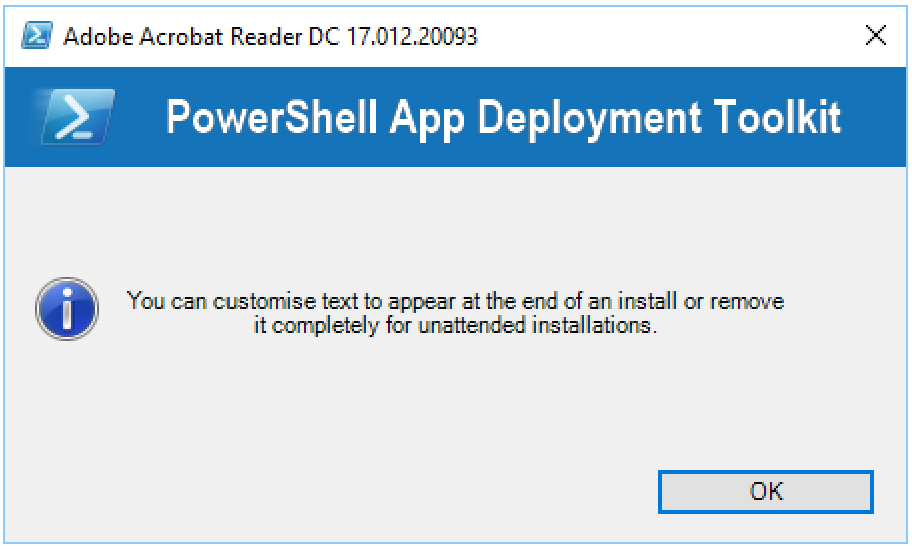

Adobe_AcrobatReaderDC_17.012.20093_EN_01

PowerShell App Deployment Toolkit

You can display custom prompts with up to 3 optional buttons and customized text.

The response selected by the user is returned as the text of the button they click.

Additionally, the prompt can be displayed asynchronously, e.g. to display a message at the end of the installation but allow the installation to return the exit code to the parent process without waiting for the user to respond to the message.

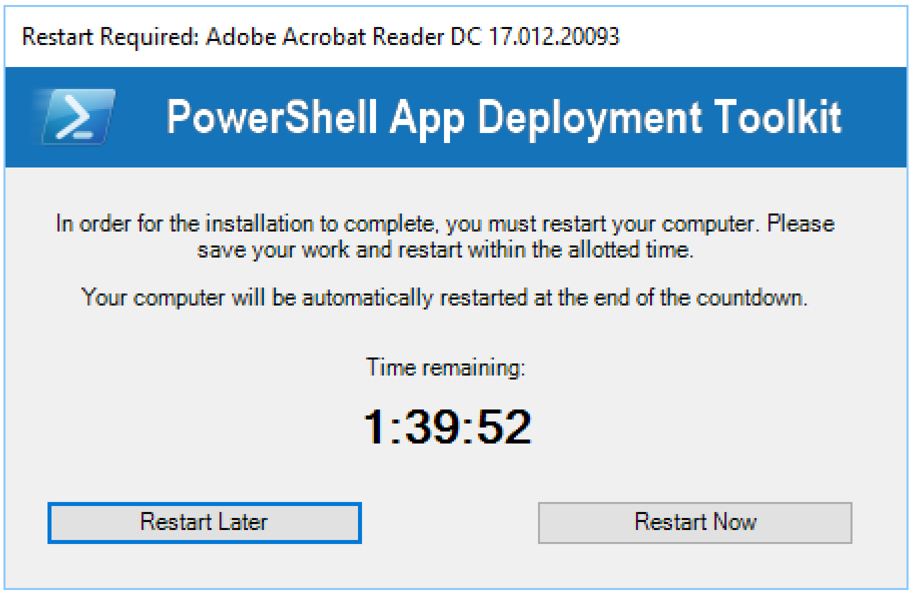

Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

You can customise text to appear at the end of an install or remove it completely for unattended installations.

Installation Restart Prompt

A restart prompt can be displayed with a countdown to automatic restart using the `Show-InstallationRestartPrompt`. Since the restart prompt is executed in a separate PowerShell session, the toolkit will still return the appropriate exit code to the parent process.



Restart Required: Adobe Acrobat Reader DC 17.012.20093

PowerShell App Deployment Toolkit

In order for the installation to complete, you must restart your computer. Please save your work and restart within the allotted time.

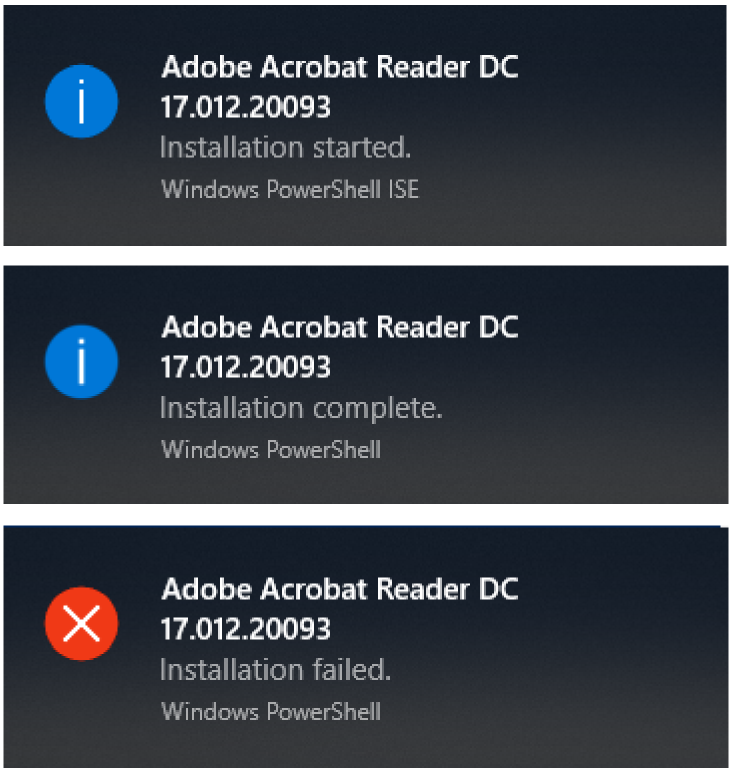
Your computer will be automatically restarted at the end of the countdown.

Time remaining:

1:39:52

Balloon Tip Notifications

Balloon tip notifications are displayed in the system tray automatically at the beginning and end of the installation. These can be turned off in the XML configuration.



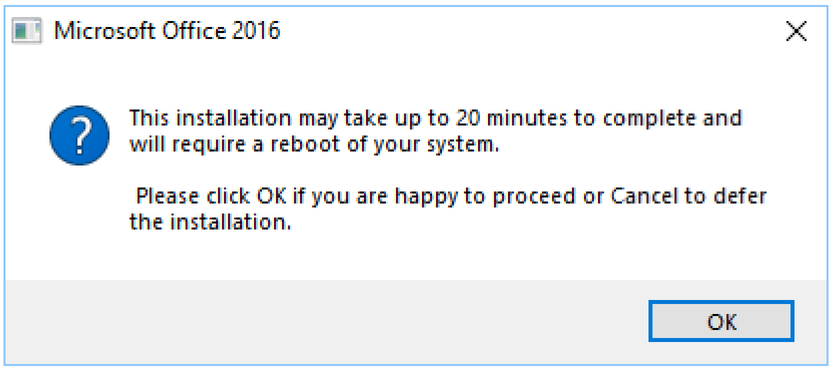
Adobe Acrobat Reader DC 17.012.20093
Installation started.
Windows PowerShell ISE

Adobe Acrobat Reader DC 17.012.20093
Installation complete.
Windows PowerShell

Adobe Acrobat Reader DC 17.012.20093
Installation failed.
Windows PowerShell

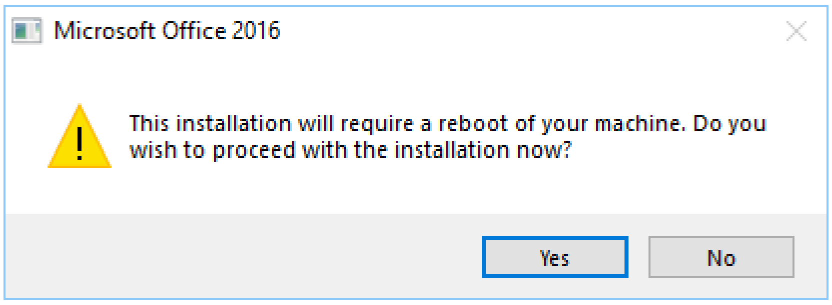
Custom Dialog box

A generic dialog box to display custom messages to the user without the toolkit branding using the function `Show-DialogBox`. This can be customized with different system icons and buttons.



This installation may take up to 20 minutes to complete and will require a reboot of your system.

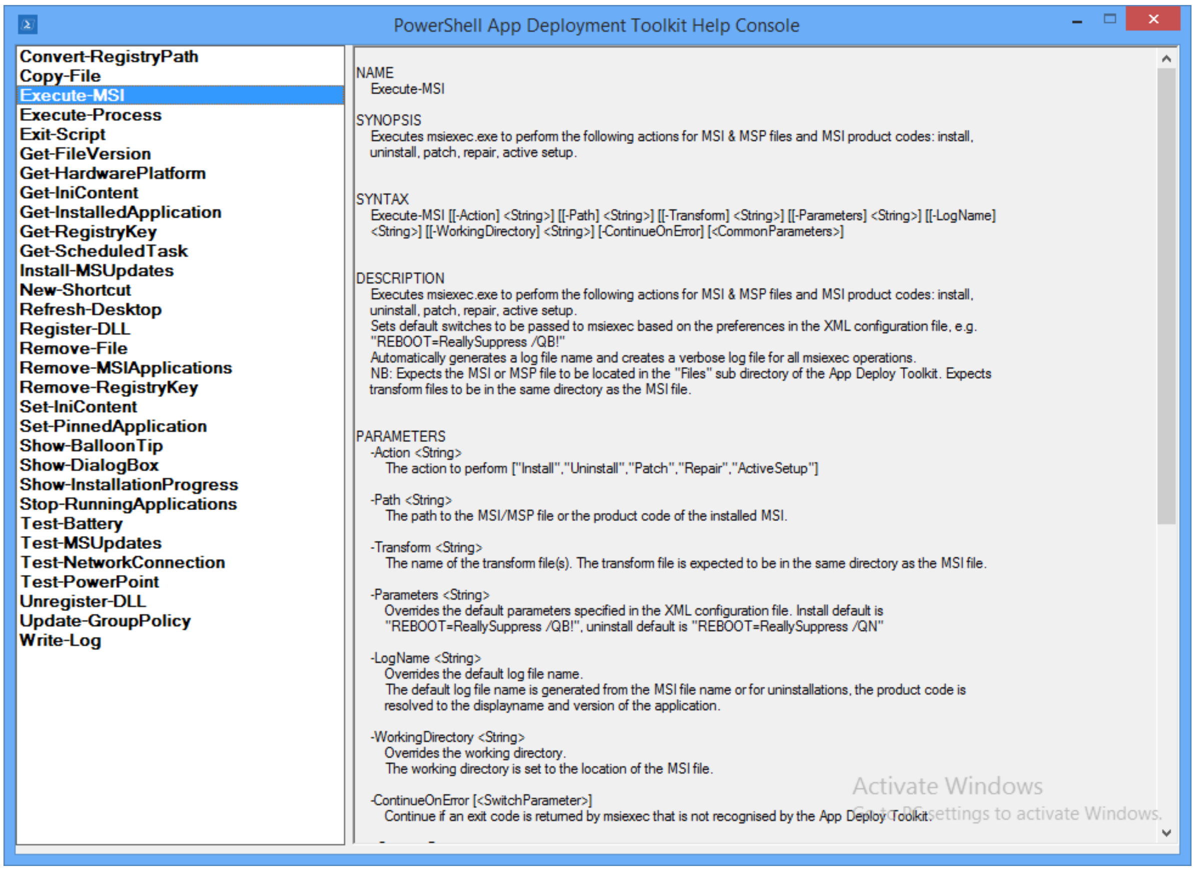
Please click OK if you are happy to proceed or Cancel to defer the installation.



This installation will require a reboot of your machine. Do you wish to proceed with the installation now?

Toolkit Help Console

The PowerShell App Deployment Toolkit Help Console. This is a separate script and can be invoked by running `AppDeployToolkitHelp.ps1`



```
NAME
   Execute-MSI

SYNOPSIS
   Executes msiexec.exe to perform the following actions for MSI & MSP files and MSI product codes: install,
   uninstall, patch, repair, active setup.


SYNTAX
   Execute-MSI [[-Action] <String>] [[-Path] <String>] [[-Transform] <String>] [[-Parameters] <String>] [[-LogName]
   <String>] [[-WorkingDirectory] <String>] [-ContinueOnError] [<CommonParameters>]


DESCRIPTION
   Executes msiexec.exe to perform the following actions for MSI & MSP files and MSI product codes: install,
   uninstall, patch, repair, active setup.
   Sets default switches to be passed to msiexec based on the preferences in the XML configuration file, e.g.
   "REBOOT=ReallySuppress /QB!"
   Automatically generates a log file name and creates a verbose log file for all msiexec operations.
   NB: Expects the MSI or MSP file to be located in the "Files" sub directory of the App Deploy Toolkit. Expects
   transform files to be in the same directory as the MSI file.


PARAMETERS
   -Action <String>
      The action to perform ["Install","Uninstall","Patch","Repair","ActiveSetup"]

   -Path <String>
      The path to the MSI/MSP file or the product code of the installed MSI.

   -Transform <String>
      The name of the transform file(s). The transform file is expected to be in the same directory as the MSI file.

   -Parameters <String>
      Overrides the default parameters specified in the XML configuration file. Install default is
      "REBOOT=ReallySuppress /QB!", uninstall default is "REBOOT=ReallySuppress /QN"

   -LogName <String>
      Overrides the default log file name.
      The default log file name is generated from the MSI file name or for uninstallations, the product code is
      resolved to the displayname and version of the application.

   -WorkingDirectory <String>
      Overrides the working directory.
      The working directory is set to the location of the MSI file.

   -ContinueOnError [<SwitchParameter>]
      Continue if an exit code is returned by msiexec that is not recognised by the App Deploy Toolkit.
```

Using The Toolkit

The *Deploy-Application.ps1* script is the only one you need to modify to deploy your application.

The script is broken down into the following sections:

Initialization - Variables such as App Vendor, App Name, App Version

Pre-Installation - Close applications, uninstall or clean-up previous versions

Installation - Install the primary application, or components of the application

Post-Installation - Drop additional files, registry tweaks

Uninstallation - Uninstall/rollback the changes performed in the install section.

Deploying an Application

There are two ways to launch the toolkit for deployment of applications.

- Launch *Deploy-Application.ps1* PowerShell script as administrator.
- Launch *Deploy-Application.exe* as administrator. This will launch the *Deploy-Application.ps1* PowerShell script without opening a PowerShell command window. Note, if the x86 PowerShell is required (for example, if CAPICOM or another x86 library is needed), launch `Deploy-Application.exe /32`.

Examples

Deploy an application for installation

```
Deploy-Application.ps1
```

Deploy an application for uninstallation in silent mode

```
Deploy-Application.ps1 -DeploymentType 'Uninstall' -DeployMode 'Silent'
```

Deploy an application for uninstallation using PowerShell x86, suppressing the PowerShell console window and deploying in silent mode.

```
Deploy-Application.exe /32 -DeploymentType 'Uninstall' -DeployMode 'Silent'
```

Deploy an application for installation, suppressing the PowerShell console window and allowing reboot codes to be returned to the parent process.

```
Deploy-Application.exe -AllowRebootPassThru
```

Deploy an application with a custom name instead of *Deploy-Application.ps1*.

```
Deploy-Application.exe 'Custom-Script.ps1'
```

Remove an application with a custom name and custom location for the script file.

```
Deploy-Application.exe -Command 'C:\Testing\Custom-Script.ps1' -DeploymentType
'Uninstall'
```

Toolkit Parameters

The following parameters are accepted by *Deploy-Application.ps1*:

DeploymentType

Specify whether to install or uninstall the application.

```
-DeploymentType 'Install' ## default
```

```
-DeploymentType 'Uninstall'
```

DeployMode

Specify the installation will be run in Interactive, Silent or NonInteractive mode:

- Interactive = Shows dialogs
- NonInteractive = Very silent, i.e. no blocking apps. This is automatically set if it is detected that the process is not running in the user session and it is not possible for anyone to provide input using a mouse or keyboard.
- Silent = No dialogs (progress and balloon tip notifications are suppressed)

```
-DeployMode 'Interactive' ## default
```

```
-DeployMode 'Silent'
```

```
-DeployMode 'NonInteractive'
```

AllowRebootPassthru

Specify whether to allow the 3010 exit code (reboot required) to be passed back to the parent process (e.g. MEMCM) if detected during an installation. If a 3010 code is passed to MEMCM, the MEMCM client will display a reboot prompt. If set to false, the 3010 return code will be replaced by a "0" (successful, no restart required).

```
-AllowRebootPassThru $true ## default
```

```
-AllowRebootPassThru $false
```

TerminalServerMode

Changes to user install mode and back to user execute mode for installing/uninstalling applications on Remote Desktop Session Host/Citrix servers

```
-TerminalServerMode $true ## default
```

```
-TerminalServerMode $false
```

DisableLogging

Disables logging to file for the script.

```
-DisableLogging
```

This is a switch parameter, so setting this enables it. When not present, the default is false.

Customizing the Toolkit

Aside from customizing the *Deploy-Application.ps1* script to deploy your application, no configuration is necessary out of the box. The following components can be configured as required:

AppDeployToolkitConfig.xml

Configure the default UI messages, MSI parameters, log file location, whether Admin rights should be required, whether log files should be compressed, log style (CMTrace or Legacy), max log size, whether debug messages should be logged, whether log entries should be written to the console, whether toolkit should re-launch as elevated logged-on console user when in SYSTEM context, whether toolkit should fall back to SYSTEM context if failure to launch toolkit as user, and whether toolkit should attempt to launch as a non-console logged on user (e.g. user logged on via terminal services) when in SYSTEM context.

AppDeployToolkitLogo.ico

To brand the balloon notifications and UI window title bars with your own custom/corporate logo, replace the *AppDeployToolkitLogo.ico* file with your own .ico file (retaining the file name)

AppDeployToolkitBanner.png

To brand the toolkit UI prompts with your own custom/corporate banner, replace the *AppDeployToolkitBanner.png* file with your own .png file (retaining the file name). The file must be in PNG format and must be 450 x 50 in size.

Logging

The toolkit generates extensive logging for all toolkit and MSI operations.

The default log directory for the toolkit and MSI log files can be specified in the XML configuration file. The default directory is *C:\Windows\Logs\Software*.

The toolkit log file is named after the application with _PSAppDeployToolkit appended to the end, e.g.

- *Oracle_JavaRuntime_1.7.0.17_EN_01_**PSAppDeployToolkit.log***

All MSI actions are logged and the log file is named according to the MSI file used on the command line, with the action appended to the log file name. For uninstallations, the MSI product code is resolved to the MSI application name and version to keep the same log file format, e.g.

- *Oracle_JavaRuntimeEnvironmentx86_1.7.0.17_EN_01_**Install.log***
- *Oracle_JavaRuntimeEnvironmentx86_1.7.0.17_EN_01_**Repair.log***
- *Oracle_JavaRuntimeEnvironmentx86_1.7.0.17_EN_01_**Patch.log***
- *Oracle_JavaRuntimeEnvironmentx86_1.7.0.17_EN_01_**Uninstall.log***

Log Compression

One of the Toolkit Options in the *AppDeployToolkitConfig.xml* file is CompressLogs. Enabling this option will create a temporary logging folder where you can save all of the log files you want to include in the single ZIP file that will be created from this folder.

To enable the CompressLogs, set the follow option in *AppDeployToolkitConfig.xml* to True:

```
<Toolkit_CompressLogs>True</Toolkit_CompressLogs>
```

When set to True, the following happens:

- Both toolkit and MSI logs are temporally placed in $envTemp$installName which gets cleaned up at the end of the install.
- At the end of the install / uninstall, the logs are compressed into a new zip file which is placed in the LogFolder location in the config file.
- The Zip file name indicates whether it is an Install / Uninstall and has the timestamp in the filename so previous logs do not get overwritten.
- If your package creates other log files, you can send them to the temporary logging folder at $envTemp$installName.

Zero-Configuration MSI Deployment

The toolkit has a zero-config MSI install feature which allows you to quickly execute an installation with zero configuration of the *Deploy-Application.ps1* file. To use this feature:

1. Place your MSI file into the *Files* directory of the toolkit. This method only support the installation of one MSI, so if more than one MSI is found, then only the first one is selected.
2. If you have an MST file, then place it into the *Files* directory of the toolkit. The MST file must have the same name as the MSI file. For example, if your MSI file name is *test01.msi*, then the MST file must be named *test01.mst*.
3. If you have any MSP files, then place it into the *Files* directory of the toolkit. You can place more than one MSP file in the folder, but you must name the files in alphabetical order to control the order in which they are installed. MSP file will be installed in alphabetical order.

Example Deployments using the Toolkit

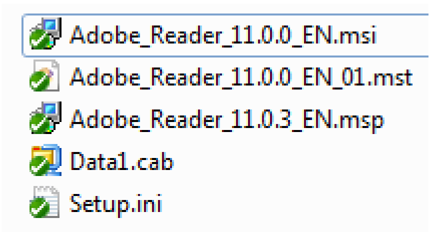
Adobe Reader - Deployment Script

In this example, we will build an Adobe Reader installation which provides the following benefits over using a standard MSI based MEMCM deployment:

- The ability to defer the installation up to 3 times
- The ability to close any applications that could cause errors during the installation
- Verification that the required disk space is available
- Full removal of any previous version of Adobe Reader (to prevent issues sometimes seen when doing an MSI upgrade, i.e. Missing previous installation source files)
- Installation of any subsequent patches required after the base MSI installation

This example is provided as a script with the toolkit, in the *Examples* folder.

- Copy the application source files in to the *Files* directory, e.g.


Adobe_Reader_11.0.0_EN.msi
Adobe_Reader_11.0.0_EN_01.mst
Adobe_Reader_11.0.3_EN.msp
Data1.cab
Setup.ini

- Customize the *Deploy-Application.ps1* script using the example code below
- Install the application by running:

```
Deploy-Application.exe
```

- Uninstall the application by running:

```
Deploy-Application.exe -DeploymentType "Uninstall"
```

Initialization

Populate these variables with the application and script details:

```
$appVendor = 'Adobe'
$appName = 'Reader'
$appVersion = '11.0.3'
$appArch = ''
$appLang = 'EN'
$appRevision = '01'
$appScriptVersion = '1.0.0'
$appScriptDate = '08/07/2013'
$appScriptAuthor = 'Your Name'
```

Pre-Install

Copy the following into the Pre-Install section:

```
## Prompt the user to close the following applications if they are running and
## allow the option to defer the installation up to 3 times:
Show-InstallationWelcome -CloseApps 'iexplore,AcroRd32,cidaemon' -AllowDefer -
DeferTimes 3

## Show Progress Message (with the default message)
Show-InstallationProgress

## Remove any previous versions of Adobe Reader
Remove-MSIApplications -Name 'Adobe Reader'
```

Installation

Copy the following into the Installation section:

```
## Install the base MSI and apply a transform
Execute-MSI -Action Install -Path 'Adobe\_Reader\_11.0.0\_EN.msi' -Transform
'Adobe\_Reader\_11.0.0\_EN\_01.mst'

## Install the patch
Execute-MSI -Action Patch -Path 'Adobe\_Reader\_11.0.3\_EN.msp'
```

Post-Installation

Copy the following into the Post-Install section:

```
## No actions required here
```

Uninstallation

Copy the following into the Uninstallation section:

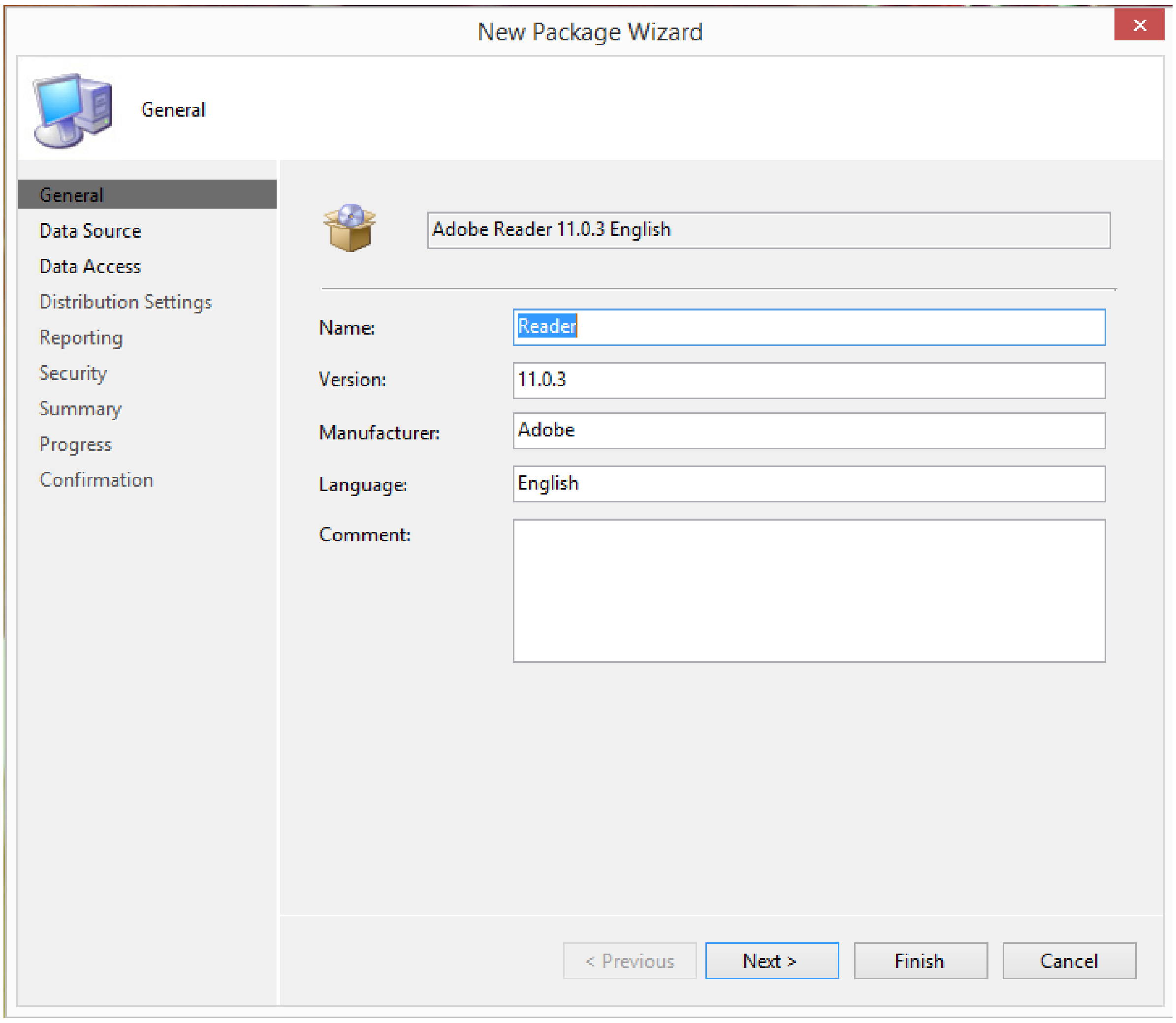
```
## Prompt the user to close the following applications if they are running:
Show-InstallationWelcome -CloseApps 'iexplore,AcroRd32,cidaemon'

## Show Progress Message (with a message to indicate the application is being
uninstalled)
Show-InstallationProgress -StatusMessage "Uninstalling Application
$installTitle. Please Wait..."

## Remove this version of Adobe Reader
Execute-MSI -Action Uninstall -Path '{AC76BA86-7AD7-1033-7B44-AB0000000001}'
```

Adobe Reader - MEMCM Package Deployment

- Copy the installation files to a network location accessible by MEMCM.
- Create a new Package:



Adobe Reader 11.0.3 English

Name: Reader

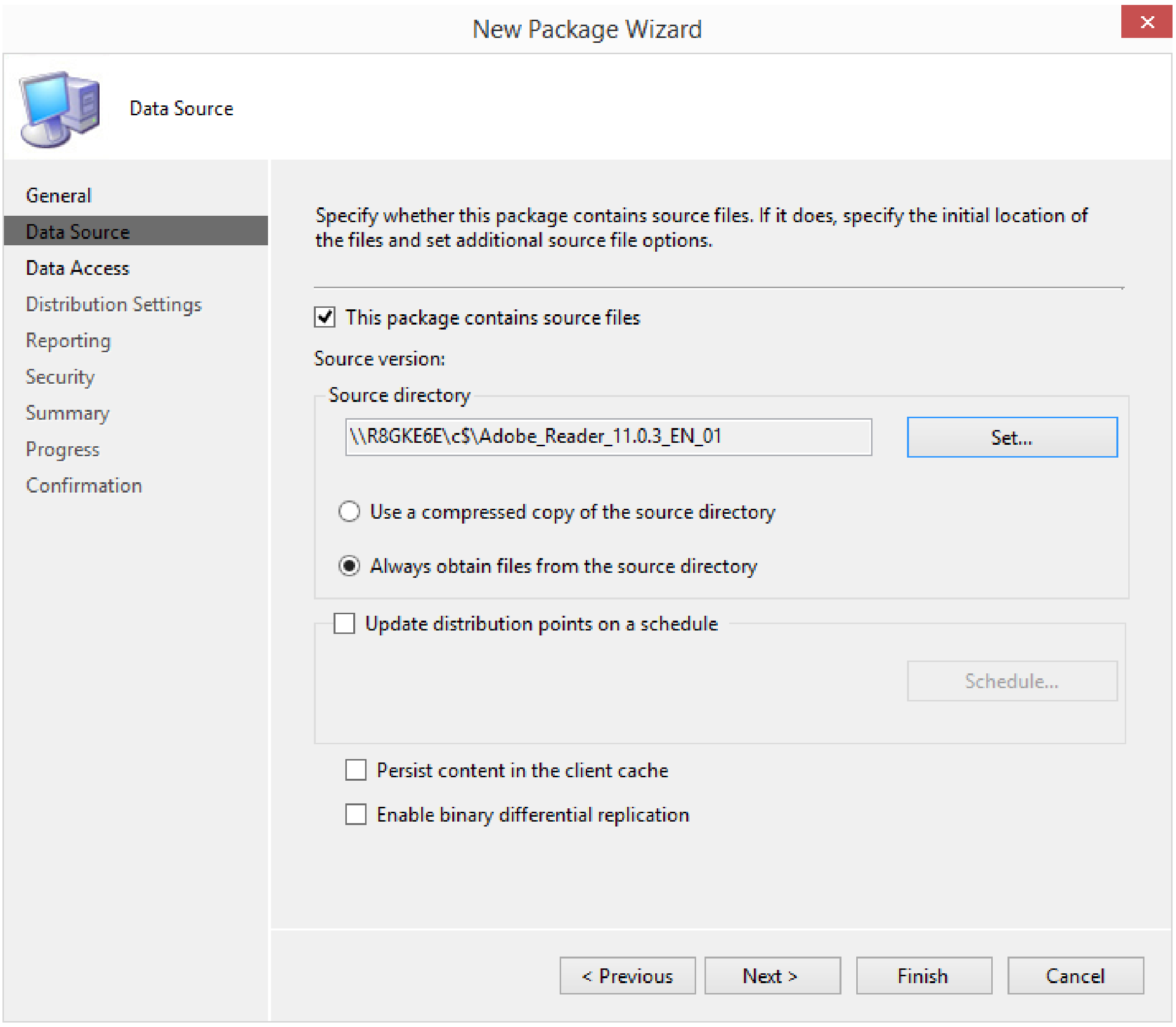
Version: 11.0.3

Manufacturer: Adobe

Language: English

Comment:

- Set the package source folder accordingly:


New Package Wizard

Data Source

General
Data Source
Data Access
Distribution Settings
Reporting
Security
Summary
Progress
Confirmation

Specify whether this package contains source files. If it does, specify the initial location of the files and set additional source file options.

[✓] This package contains source files

Source version:

Source directory

\\R8GKE6E\c$\Adobe_Reader_11.0.3_EN_01

Set...

○ Use a compressed copy of the source directory

● Always obtain files from the source directory

☐ Update distribution points on a schedule

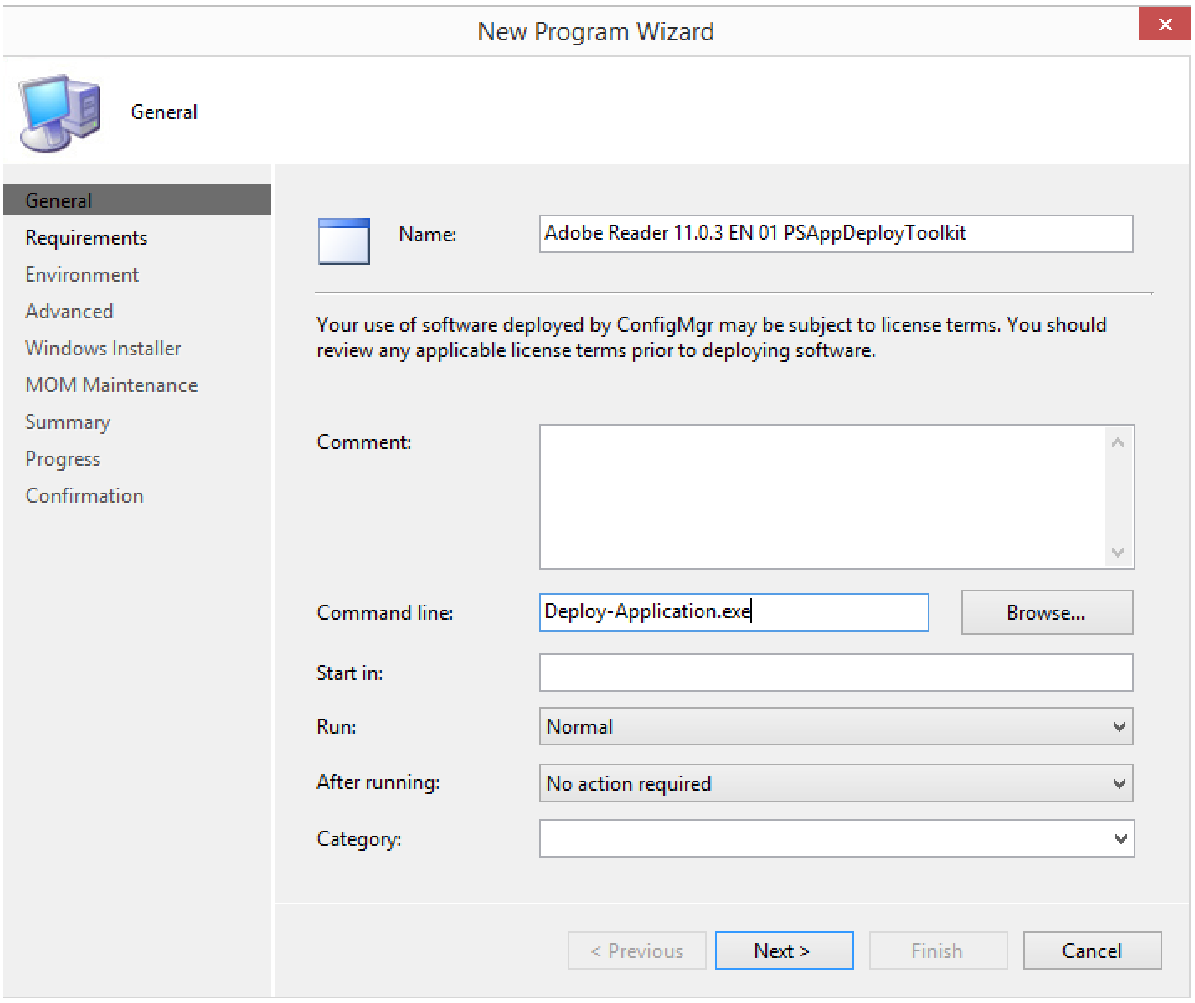
Schedule...

☐ Persist content in the client cache

☐ Enable binary differential replication

< Previous | Next > | Finish | Cancel

- Accept the defaults for the rest of the package (or modify according to your environment)
- Distribute the content of the package to the relevant Distribution Points
- Create a new Program for the package:



Name: Adobe Reader 11.0.3 EN 01 PSAppDeployToolkit

Your use of software deployed by ConfigMgr may be subject to license terms. You should review any applicable license terms prior to deploying software.

Comment:

Command line: Deploy-Application.exe

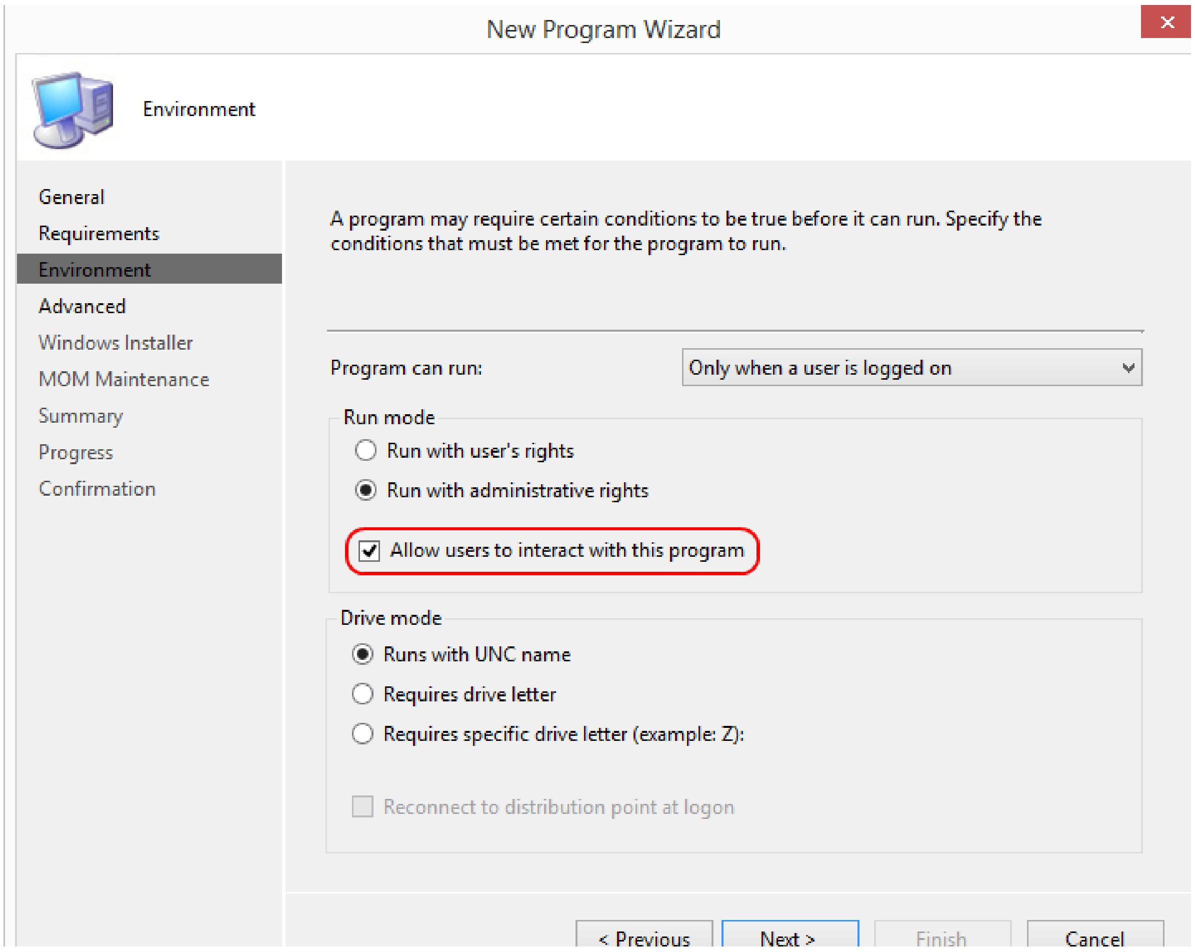
Start in:

Run: Normal

After running: No action required

Category:

- Accept the defaults for the requirements of the program (or modify according to your environment)
- On the Environment page, ensure you use a combination of settings that allows the user to interact with the application. Failure to do so will result in the application installing silently:


New Program Wizard

Environment

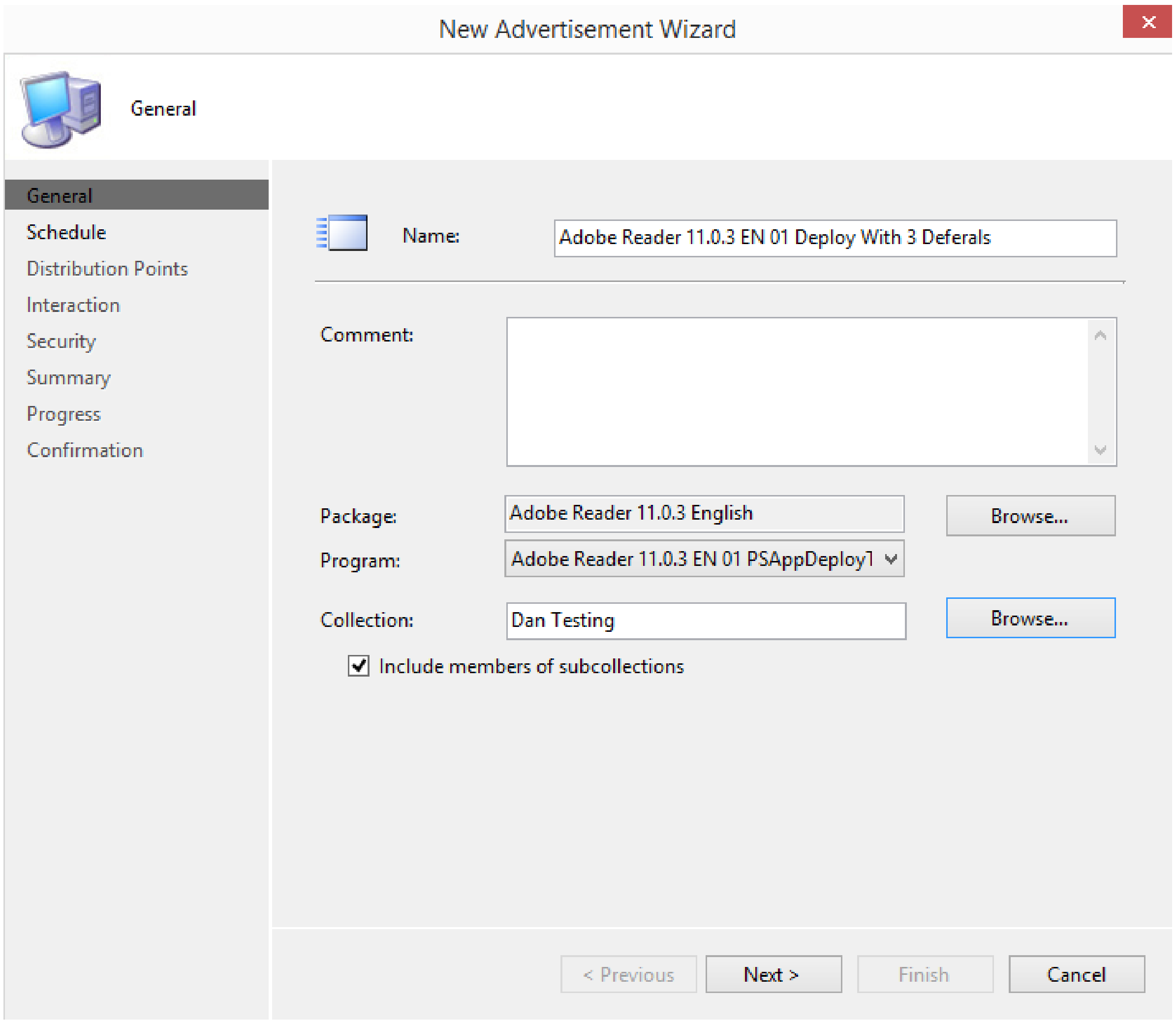
A program may require certain conditions to be true before it can run. Specify the conditions that must be met for the program to run.

Program can run: Only when a user is logged on

Run mode
- ○ Run with user's rights
- ● Run with administrative rights
- ☑ Allow users to interact with this program

Drive mode
- ● Runs with UNC name
- ○ Requires drive letter
- ○ Requires specific drive letter (example: Z):
- ☐ Reconnect to distribution point at logon

- Accept the defaults for the rest of the program (or modify according to your environment)
- Create a new Advertisement for the Package and set your target collection accordingly:


New Advertisement Wizard

General

Name: Adobe Reader 11.0.3 EN 01 Deploy With 3 Deferals

Comment:

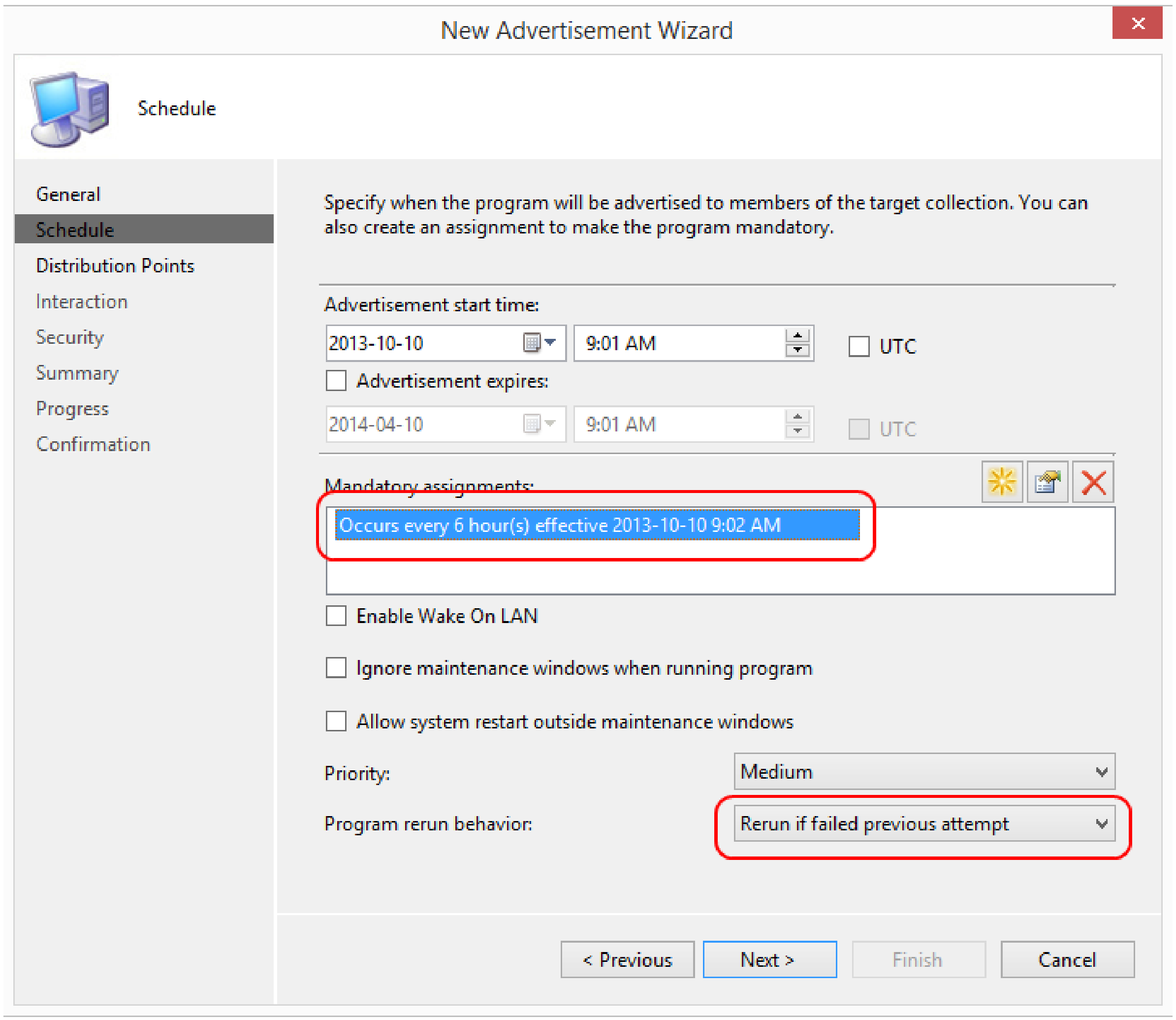
Package: Adobe Reader 11.0.3 English

Program: Adobe Reader 11.0.3 EN 01 PSAppDeploy1

Collection: Dan Testing

☑ Include members of subcollections

- Set a recurring schedule for the Mandatory Assignment. This dictates how frequently the application should attempt to install. Additionally, ensure that "Rerun if failed previous attempt" is enabled. These settings are required when using the deferral system and ensure that if a user defers the install, the install will retry after the specified interval:


New Advertisement Wizard

Schedule

Specify when the program will be advertised to members of the target collection. You can also create an assignment to make the program mandatory.

Advertisement start time: 2013-10-10 9:01 AM ☐ UTC

☐ Advertisement expires: 2014-04-10 9:01 AM ☐ UTC

Mandatory assignments:

Occurs every 6 hour(s) effective 2013-10-10 9:02 AM

☐ Enable Wake On LAN

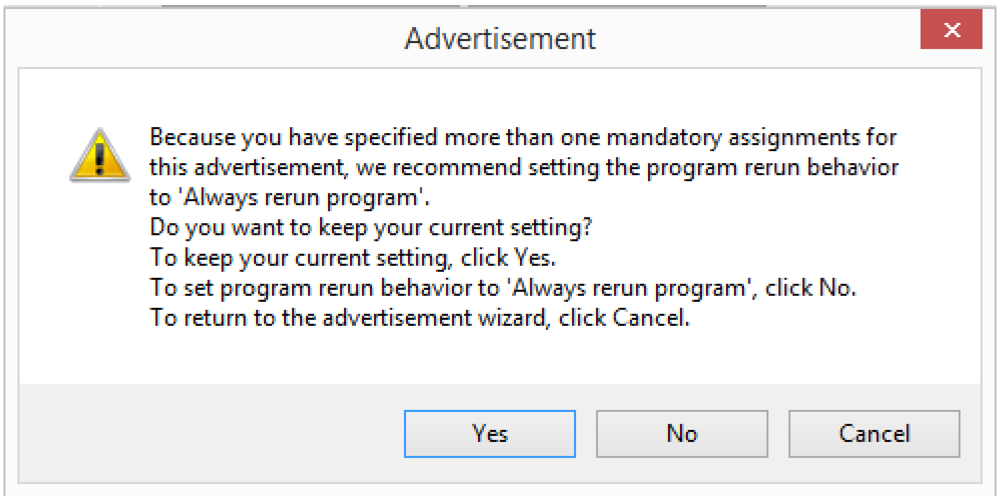
☐ Ignore maintenance windows when running program

☐ Allow system restart outside maintenance windows

Priority: Medium

Program rerun behavior: Rerun if failed previous attempt

- When prompted with the following dialog box, select Yes:

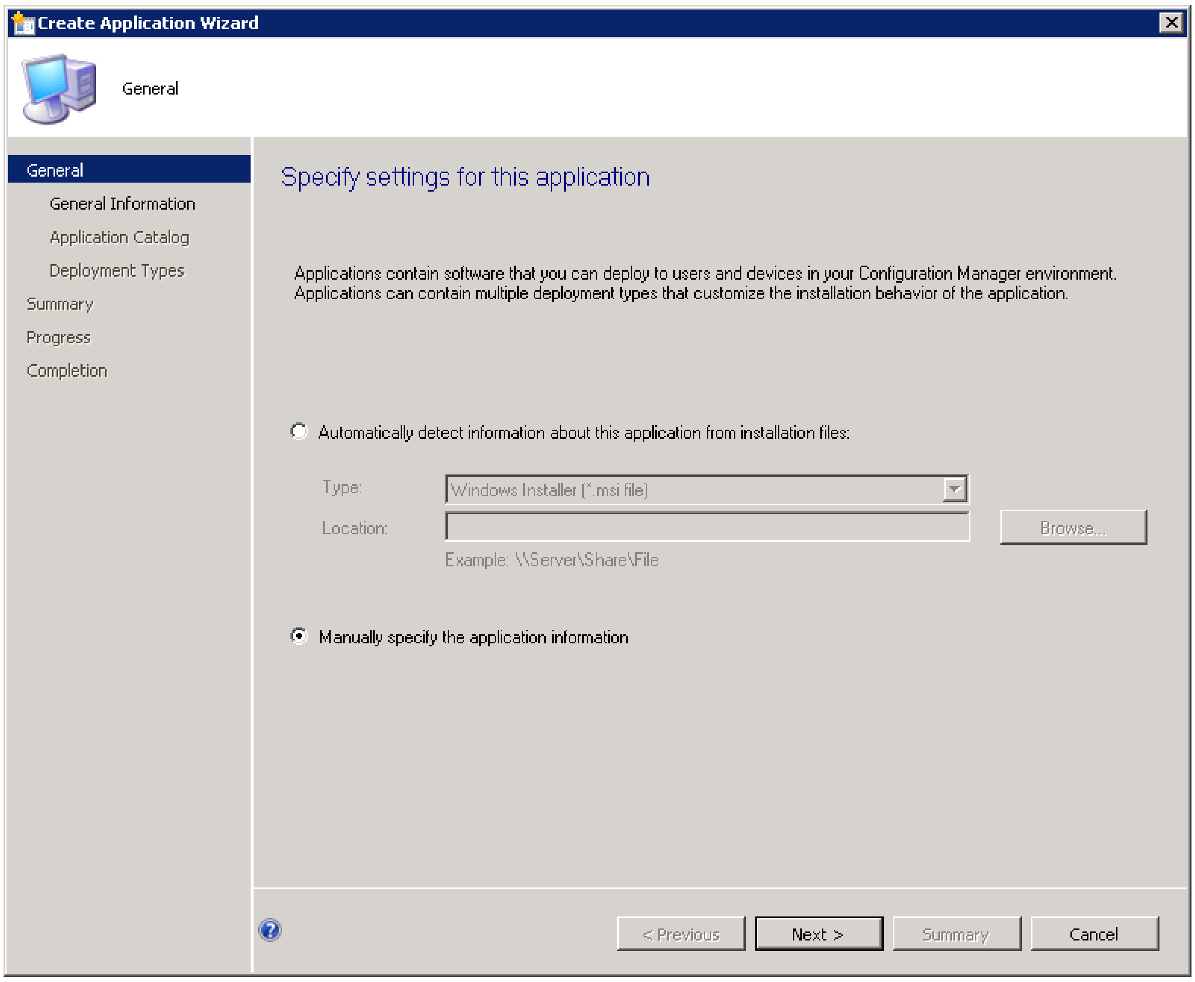

Advertisement

Because you have specified more than one mandatory assignments for this advertisement, we recommend setting the program rerun behavior to 'Always rerun program'.
Do you want to keep your current setting?
To keep your current setting, click Yes.
To set program rerun behavior to 'Always rerun program', click No.
To return to the advertisement wizard, click Cancel.

- Accept the defaults for the rest of the advertisement (or modify according to your environment). The deployment should start on your target machines shortly.

Adobe Reader - MEMCM Application Model Deployment

- Copy the installation files to a network location accessible by MEMCM.
- Create a new Application and manually specify the application information:



Create Application Wizard

General

Specify settings for this application

Applications contain software that you can deploy to users and devices in your Configuration Manager environment. Applications can contain multiple deployment types that customize the installation behavior of the application.

( ) Automatically detect information about this application from installation files:

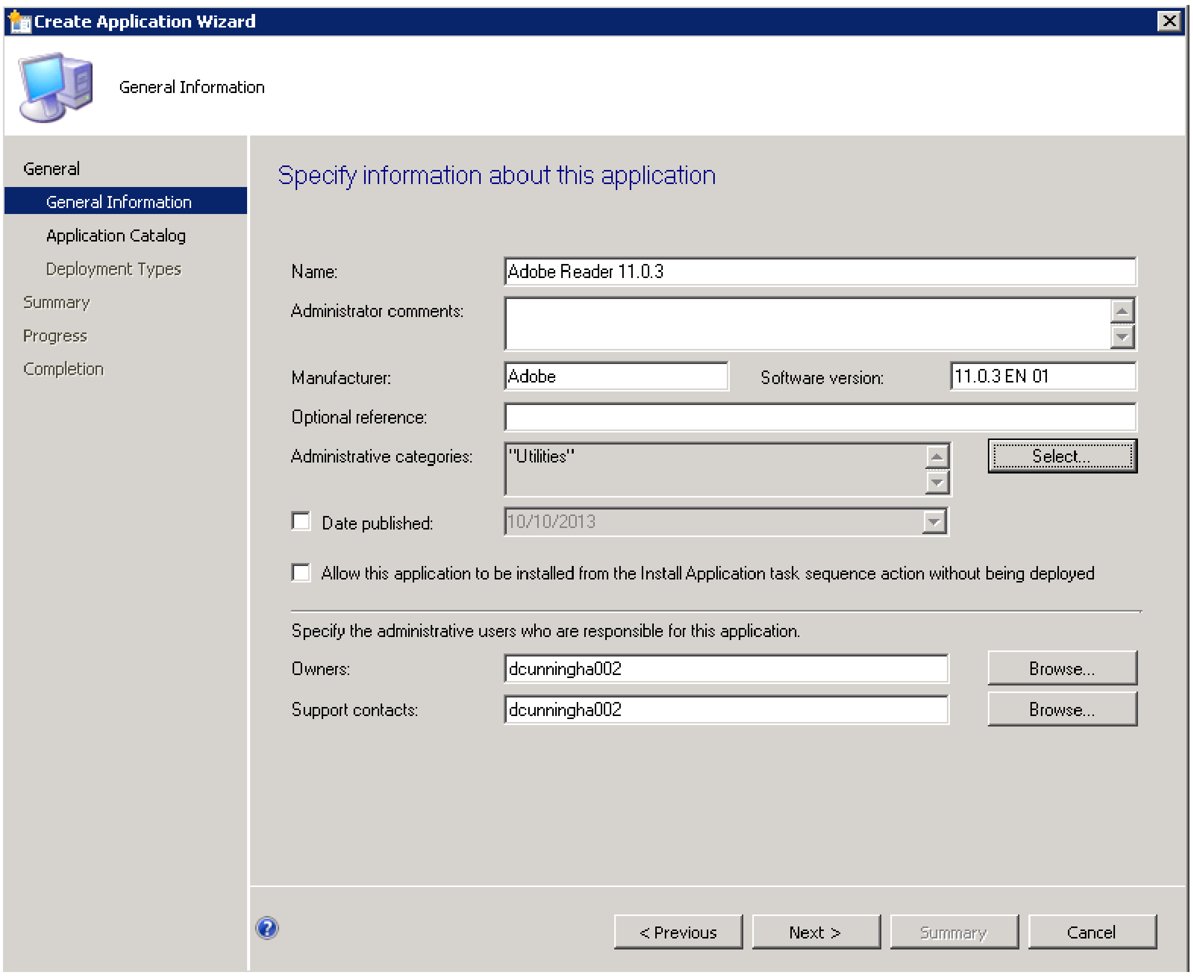
Type: Windows Installer (*.msi file)

Location:

Example: \\Server\Share\File

(✓) Manually specify the application information

- Populate the application details accordingly:

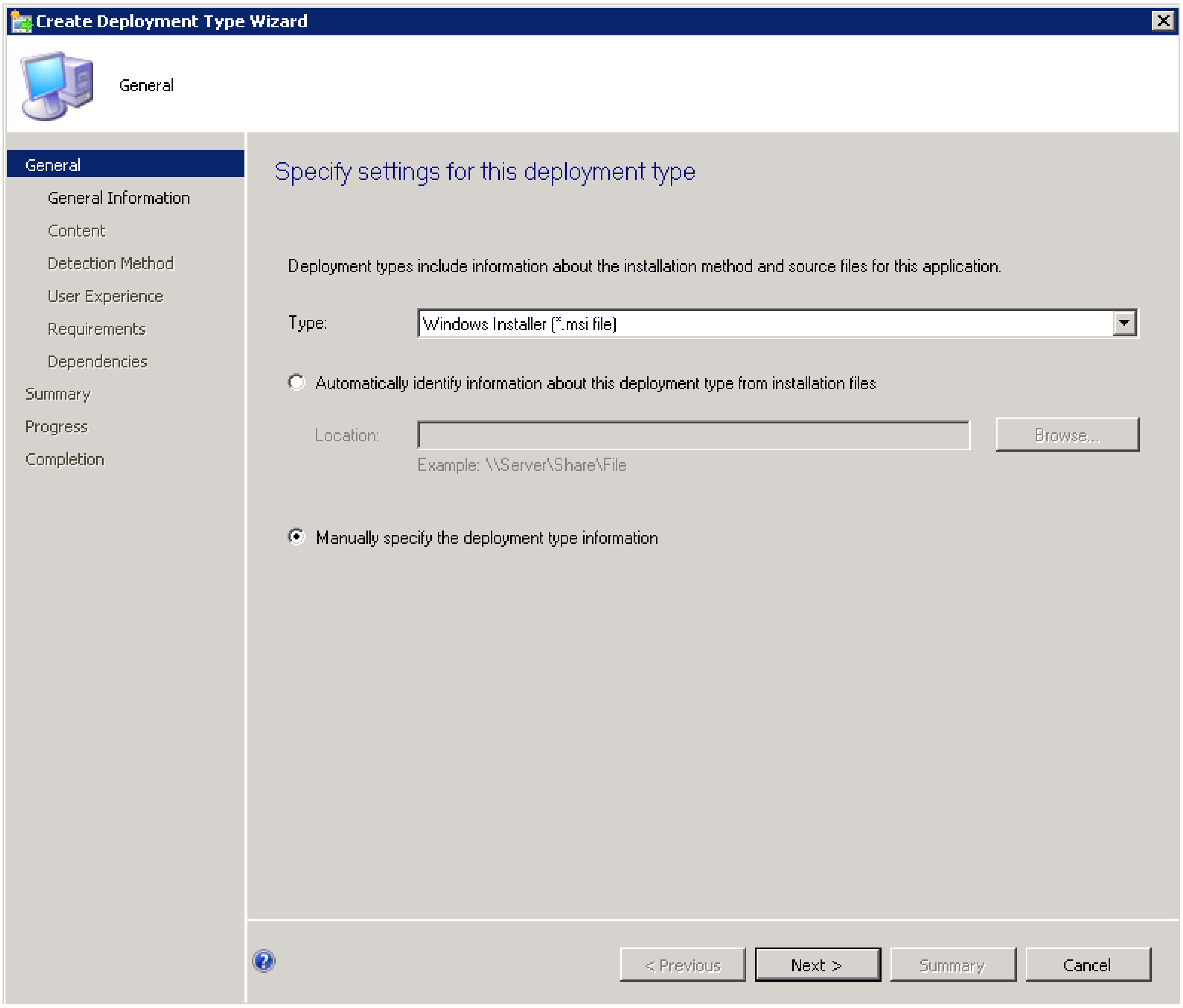


Specify information about this application

Name: Adobe Reader 11.0.3
Administrator comments:
Manufacturer: Adobe Software version: 11.0.3 EN 01
Optional reference:
Administrative categories: "Utilities"
☐ Date published: 10/10/2013
☐ Allow this application to be installed from the Install Application task sequence action without being deployed

Specify the administrative users who are responsible for this application.
Owners: dcunningha002
Support contacts: dcunningha002

- Populate the application catalog details if required
- Add a new Deployment Type and manually specify the deployment type information:



Specify settings for this deployment type

Deployment types include information about the installation method and source files for this application.

Type: Windows Installer (*.msi file)

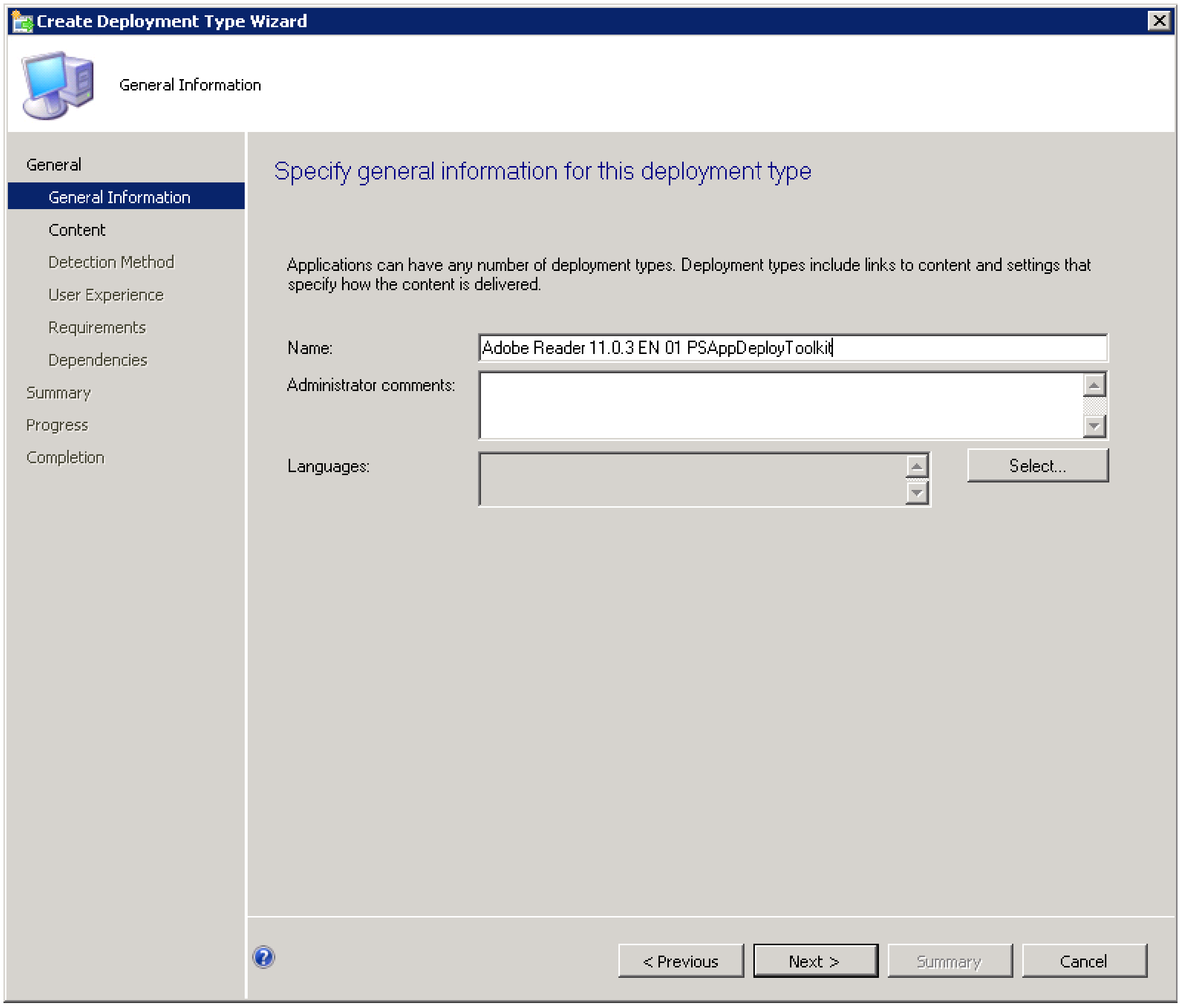
( ) Automatically identify information about this deployment type from installation files

Location:

Example: \\Server\Share\File

(●) Manually specify the deployment type information

- Populate the deployment type details accordingly:



Specify general information for this deployment type

Applications can have any number of deployment types. Deployment types include links to content and settings that specify how the content is delivered.

Name: Adobe Reader 11.0.3 EN 01 PSAppDeployToolkit

Administrator comments:

Languages:

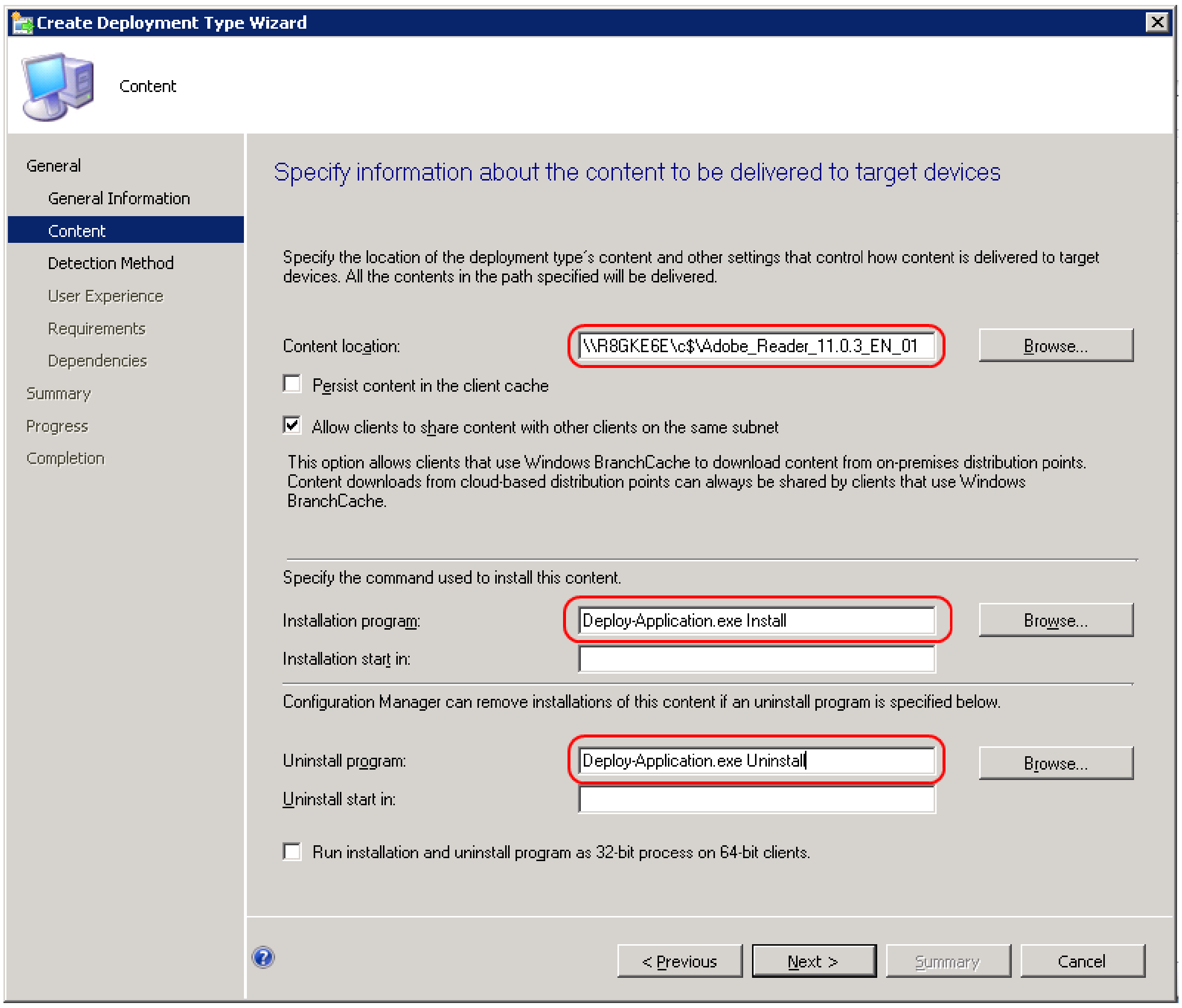
- Set the content location. Additionally, set the Install and Uninstall programs accordingly. They should be:

```
Deploy-Application.exe -DeploymentType "Install"
```

And:

```
Deploy-Application.exe -DeploymentType "Uninstall"
```

respectively.



Specify information about the content to be delivered to target devices

Specify the location of the deployment type's content and other settings that control how content is delivered to target devices. All the contents in the path specified will be delivered.

Content location: \\R8GKE6E\c$\Adobe_Reader_11.0.3_EN_01

☐ Persist content in the client cache

☑ Allow clients to share content with other clients on the same subnet

This option allows clients that use Windows BranchCache to download content from on-premises distribution points. Content downloads from cloud-based distribution points can always be shared by clients that use Windows BranchCache.

Specify the command used to install this content.

Installation program: Deploy-Application.exe Install

Installation start in:

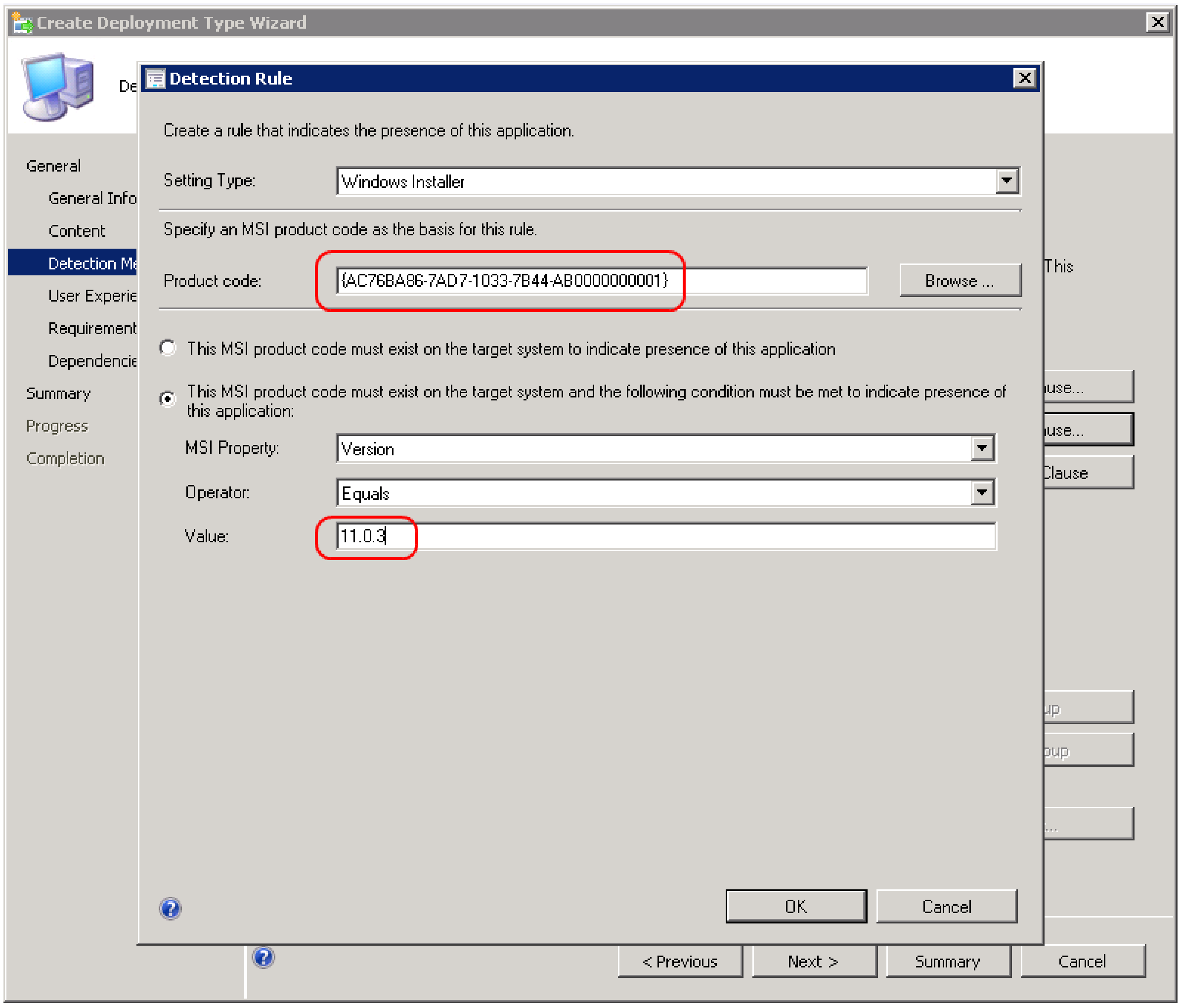
Configuration Manager can remove installations of this content if an uninstall program is specified below.

Uninstall program: Deploy-Application.exe Uninstall

Uninstall start in:

☐ Run installation and uninstall program as 32-bit process on 64-bit clients.

- Create a new detection rule. Specify the base MSI product code and modify the Version to be the same as the final version after all patches are installed:


Detection Rule

Create a rule that indicates the presence of this application.

Setting Type: Windows Installer

Specify an MSI product code as the basis for this rule.

Product code: {AC76BA86-7AD7-1033-7B44-AB0000000001}

This MSI product code must exist on the target system to indicate presence of this application

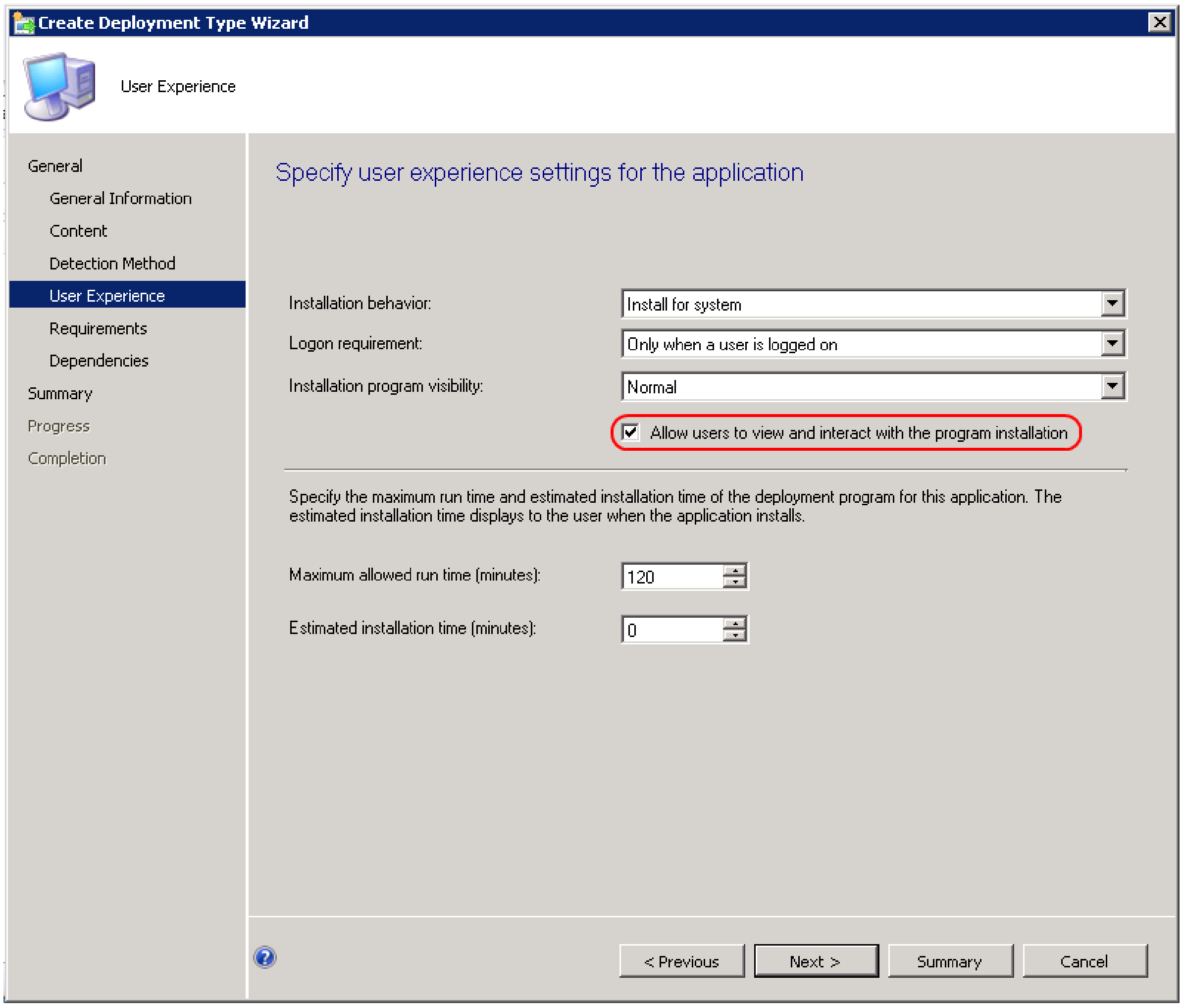
This MSI product code must exist on the target system and the following condition must be met to indicate presence of this application:

MSI Property: Version

Operator: Equals

Value: 11.0.3

- On the User Experience page, ensure you use a combination of settings that allows the user to interact with the application. Failure to do so will result in the application installing silently:



Specify user experience settings for the application

Installation behavior: Install for system

Logon requirement: Only when a user is logged on

Installation program visibility: Normal

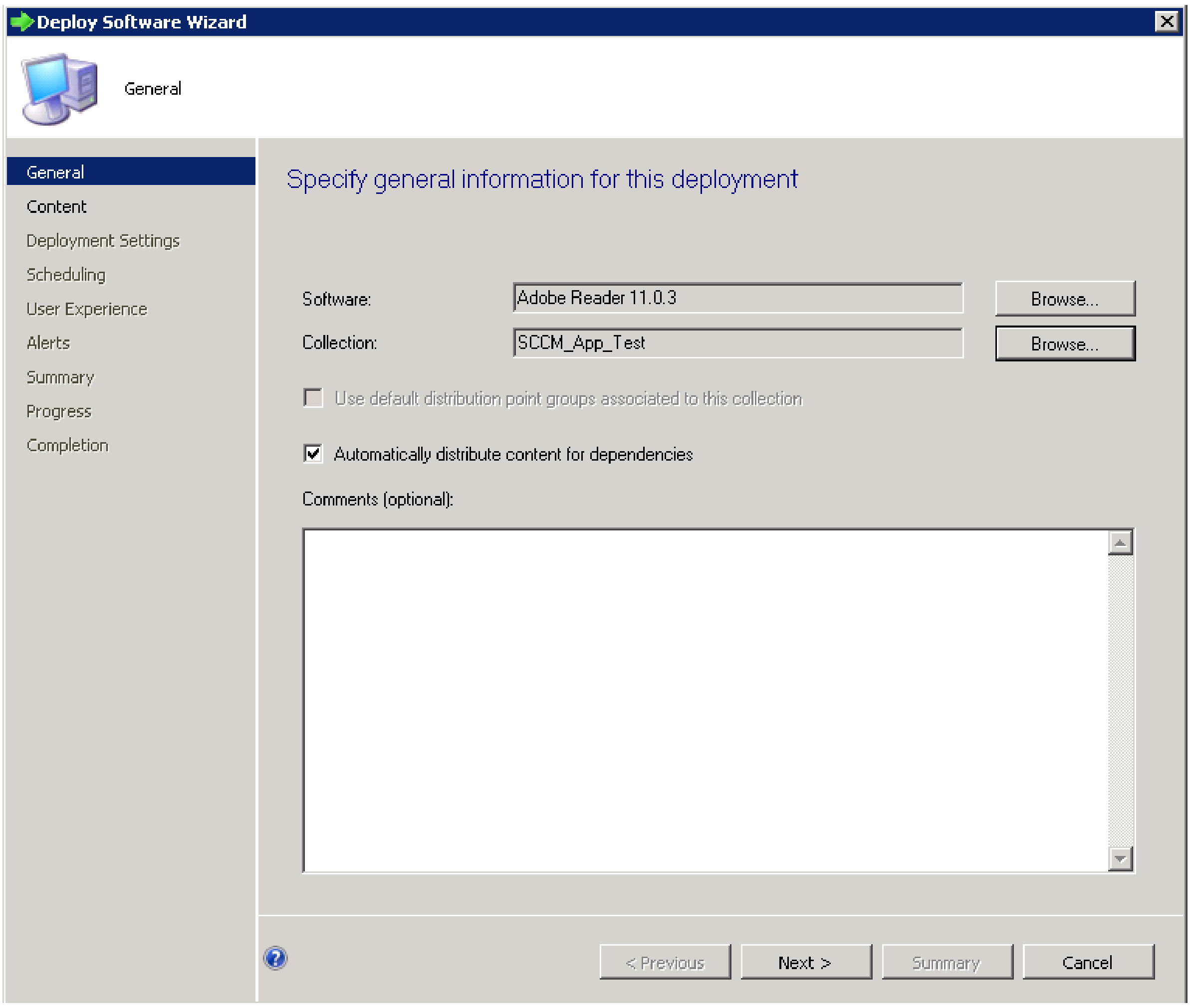
☑ Allow users to view and interact with the program installation

Specify the maximum run time and estimated installation time of the deployment program for this application. The estimated installation time displays to the user when the application installs.

Maximum allowed run time (minutes): 120

Estimated installation time (minutes): 0

- Leave the requirements page blank (or modify according to your environment)
- Leave the software dependencies page blank (or modify according to your environment)
- Accept the defaults to create the Application
- Deploy the Application:


Deploy Software Wizard

General

Specify general information for this deployment

Software: Adobe Reader 11.0.3

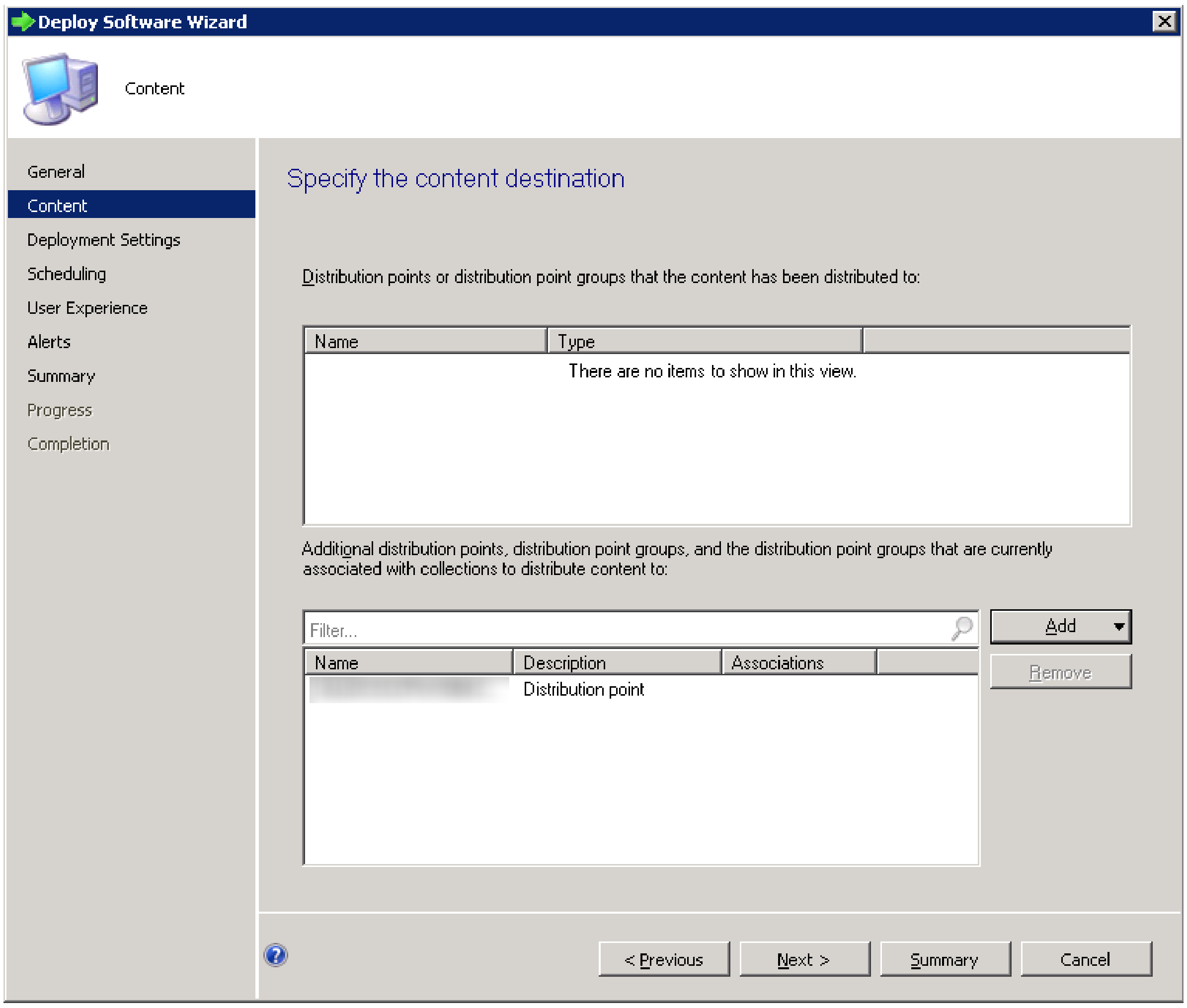
Collection: SCCM_App_Test

☐ Use default distribution point groups associated to this collection

☑ Automatically distribute content for dependencies

Comments (optional):

- Select the relevant Distribution Points:


Deploy Software Wizard

Content

Specify the content destination

Distribution points or distribution point groups that the content has been distributed to:

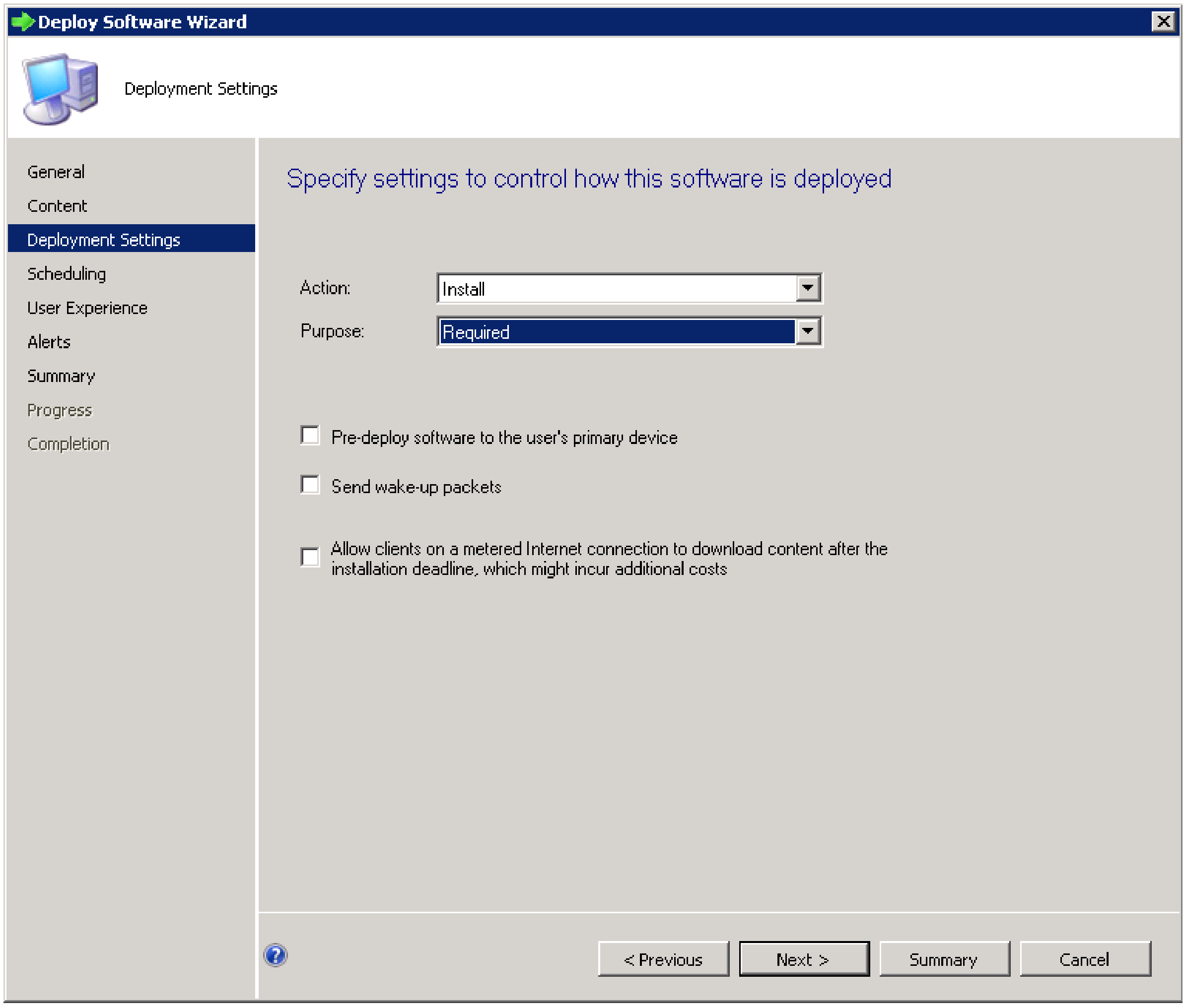
Name	Type
There are no items to show in this view.	

Additional distribution points, distribution point groups, and the distribution point groups that are currently associated with collections to distribute content to:

Filter...

Name	Description	Associations
	Distribution point	

- Configure deployment settings according to whether it should be a mandatory or app catalogue based deployment:



Specify settings to control how this software is deployed

Action: Install

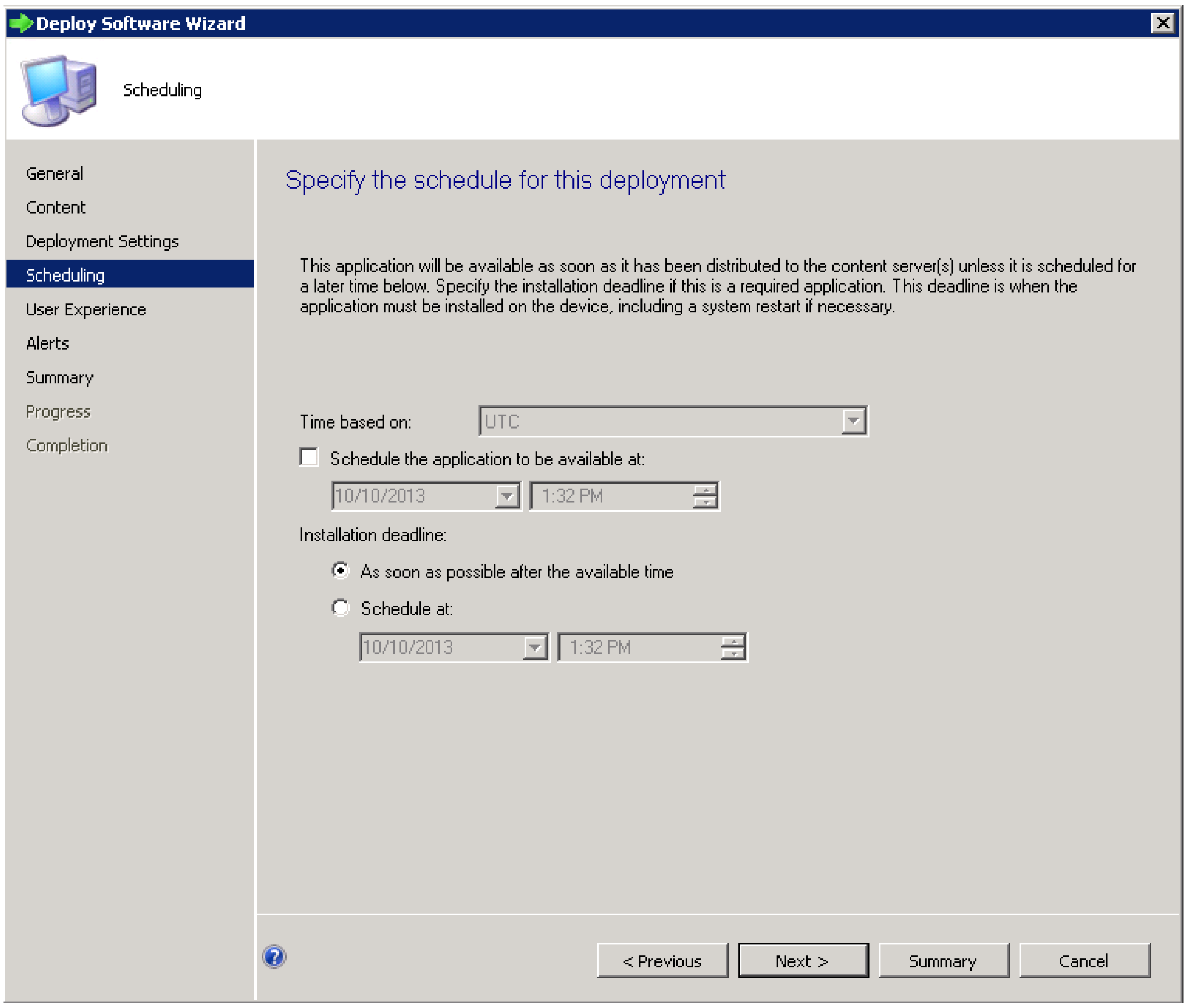
Purpose: Required

☐ Pre-deploy software to the user's primary device

☐ Send wake-up packets

☐ Allow clients on a metered Internet connection to download content after the installation deadline, which might incur additional costs

- Specify the deployment schedule:



Specify the schedule for this deployment

This application will be available as soon as it has been distributed to the content server(s) unless it is scheduled for a later time below. Specify the installation deadline if this is a required application. This deadline is when the application must be installed on the device, including a system restart if necessary.

Time based on: UTC

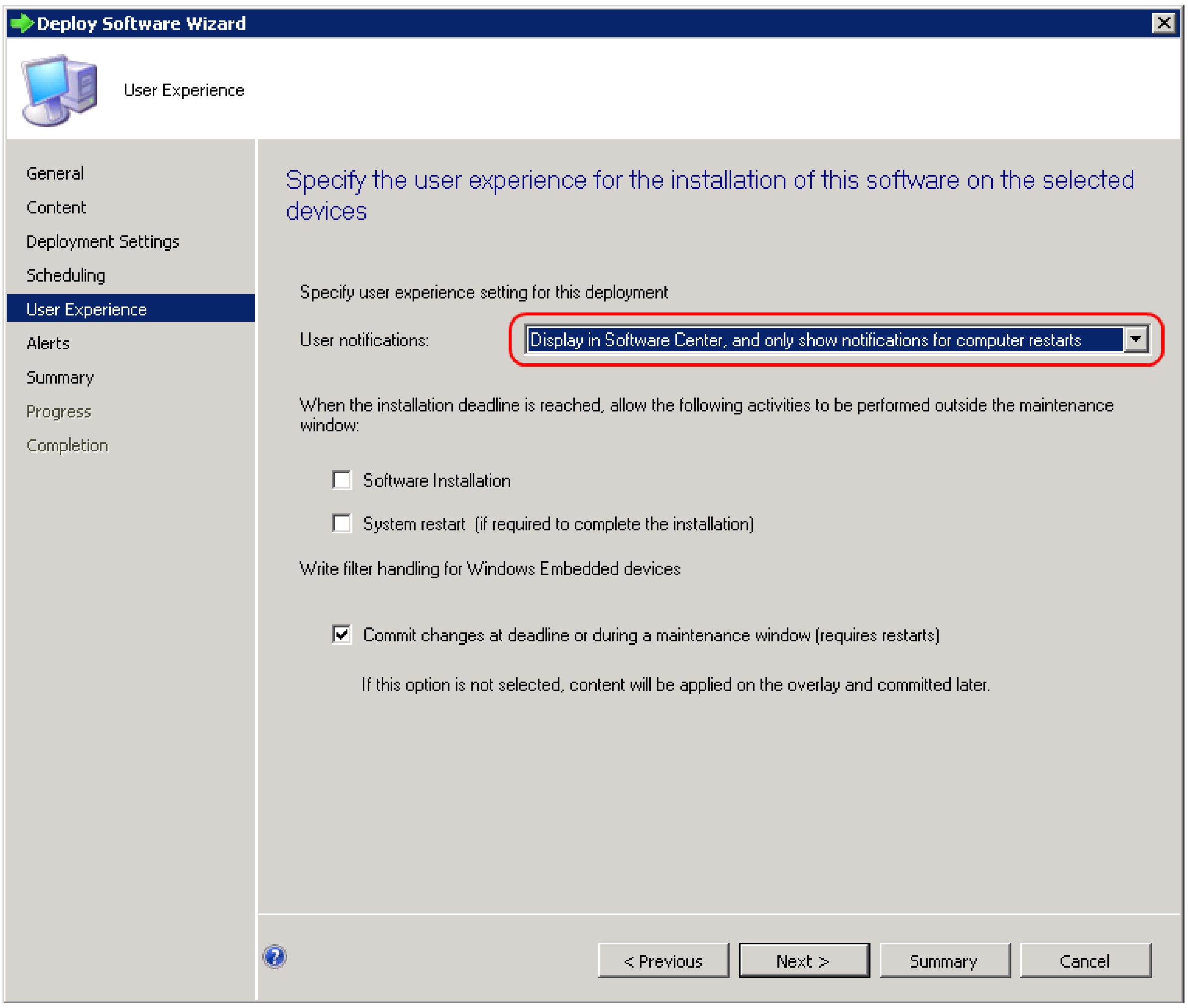
☐ Schedule the application to be available at: 10/10/2013 1:32 PM

Installation deadline:

◉ As soon as possible after the available time

○ Schedule at: 10/10/2013 1:32 PM

- Specify User notification settings. In order to prevent excess noise, we recommend only showing notifications for computer restarts:



Specify the user experience for the installation of this software on the selected devices

Specify user experience setting for this deployment

User notifications: Display in Software Center, and only show notifications for computer restarts

When the installation deadline is reached, allow the following activities to be performed outside the maintenance window:

☐ Software Installation

☐ System restart (if required to complete the installation)

Write filter handling for Windows Embedded devices

☑ Commit changes at deadline or during a maintenance window (requires restarts)

If this option is not selected, content will be applied on the overlay and committed later.

- Accept the defaults for the rest of the Deployment (or modify according to your environment)

Important note regarding MEMCM Application Model and Deferrals

The MEMCM Application Model does not have the flexibility to schedule Mandatory Assignments on a recurring schedule like MEMCM packages do. Instead, this is determined by the frequency of Software Deployment evaluation cycle in the MEMCM Agent Custom Settings. You can modify this to reduce the time from the default of once a day, however this may increase the load on your MEMCM servers and clients, and is not configurable on a per application basis:

Office 2013 SP1 - Advanced Deployment Script

This example is provided as a script with the toolkit, in the *Examples* folder. This provides a number of benefits over the standard Microsoft Office Setup Bootstrapper:

- A component based architecture so that core products can be installed, and subsequent components can be installed using the same package with different command-line switches
- The ability to defer the installation up to 3 times
- The ability to close any applications that could cause errors during the installation
- Verification that the required disk space is available
- Full removal of any previous version of Microsoft Office 2007, 2010 or 2013
- Installation of any subsequent patches required after the base installation
- Activation of Microsoft Office components

Note: Office requires a number of modifications in order to install. Please refer to Microsoft's documentation on configuration. This installation script tries to take a lot of work out of the process for you, but you still need to know what you're doing in order to set it up correctly.

The folder structure is laid out as follows:

- *Files*
    - Office installation files should be placed here
        - Office Configuration MSP created with the Office Customisation Tool should be placed in the *Config* subfolder and be named *Office2013ProPlus.MSP*. Modify the script accordingly if you wish to change. For a basic MSP, you should probably configure Access, Word, Excel and PowerPoint to be the only core applications to install. We can add everything else as components.
        - Customised *Config.xml* file should be edited in *ProPlus.WW* subfolder. At a minimum, you should modify the settings as follows:
            - Display Level = "none"
            - CompletionNotice = "no"
            - SuppressModal = "yes"
            - NoCancel = "yes"
            - AcceptEula = "yes"
        - Security updates and service pack extracted MSPs should be placed in the *Updates* subfolder.
- *SupportFiles*
    - Contains custom *Config.XML* files which are used to add specific components that might be considered unnecessary in a standard Office install, but could be added later using command-line switches
    - Contains Office Scrub tools for Office 2007, 2010 and 2013

Once the folder structure is laid out correctly and the custom *Deploy-Application.ps1* is added (as well as the AppDeployToolkit files themselves), the following command-lines are valid:

- Installs Office 2013 with core products

```
Deploy-Application.exe
```

- Installs Office 2013 with core products and InfoPath

```
Deploy-Application.exe -AddInfoPath
```

- Installs InfoPath to an existing Office 2013 installation

```
Deploy-Application.exe -AddComponentsOnly -AddInfoPath
```

Reference - Toolkit Exit Codes

The toolkit has a number of internal exit codes for any issues that may occur.

Exit Code	Description
60000-68999	Reserved for built-in exit codes in *Deploy-Application.ps1*, *Deploy-Application.exe*, and *AppDeployToolkitMain.ps1*.
69000 - 69999	Recommended for user customized exit codes in *Deploy-Application.ps1*.
70000 - 79999	Recommended for user customized exit codes in *Deploy-Application.ps1*.
60001	An error occurred in *Deploy-Application.ps1*. Check your script syntax use.
60002	Administrator privileges required for `Execute-ProcessAsUser` function.
60003	Failure when loading .NET Winforms / WPF Assemblies.
60004	Failure when displaying the Blocked Application dialog.
60005	AllowSystemInteractionFallback option was not selected in the config XML file, so toolkit will not fall back to SYSTEM context with no interaction.
60006	Failed to export the schedule task XML file in `Execute-ProcessAsUser` function.
60007	*Deploy-Application.ps1* failed to dot source *AppDeployToolkitMain.ps1* either because it could not be found or there was an error while it was being dot sourced.
60008	The -UserName parameter in the `Execute-ProcessAsUser` function has a default value that is empty because no logged in users were detected when the toolkit was launched.
60009	*Deploy-Application.exe* failed before *PowerShell.exe* process could be launched.
60013	If `Execute-Process` function captures an exit code out of range for int32 then return this custom exit code.

Reference - Toolkit Variables

The toolkit has a number of internal variables which can be used in your script. Outlined below are each of them:

Variable	Description
Toolkit Name	
$appDeployToolkitName	Short-name of toolkit without spaces
$appDeployMainScriptFriendlyName	Full name of toolkit including spaces
Script Info	
$appDeployMainScriptVersion	Version number of the toolkit
$appDeployMainScriptMinimumConfigVersion	Minimum version of the config XML file required by the toolkit
$appDeployMainScriptDate	Date toolkit was last modified
$appDeployMainScriptParameters	Contains all parameters and values specified when toolkit was launched
Datetime and Culture	
$currentDateTime	Current date & time when the toolkit was launched
$currentTime	Current time when toolkit was launched
$currentDate	Current date when toolkit was launched
$currentTimeZoneBias	TimeZone bias based on the current date/time
$culture	Object which contains all of the current Windows culture settings
$currentLanguage	Current Windows two letter ISO language name (e.g. EN, FR, DE, JA etc)
$currentUILanguage	Current Windows two letter UI ISO language name (e.g. EN, FR, DE, JA etc)
Environment Variables	***(path examples are for Windows 7 and higher)***
$envHost	Object that contains details about the current PowerShell console
$envShellFolders	Object that contains properties from registry path: *HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders*
$envAllUsersProfile	%ALLUSERSPROFILE% e.g. *C:\ProgramData*
$envAppData	%APPDATA% e.g. *C:\Users\%USERNAME%\AppData\Roaming*
$envArchitecture	%PROCESSOR_ARCHITECTURE% e.g. AMD64/IA64/x86 - This doesn't tell you the architecture of the processor but only of the current process, so it returns "x86" for a 32-bit WOW process running on 64-bit Windows.
$envCommonProgramFiles	%COMMONPROGRAMFILES% e.g. *C:\Program Files\Common Files*)
$envCommonProgramFilesX86	%COMMONPROGRAMFILES(x86)% e.g. *C:\Program Files (x86)\Common Files*
$envCommonDesktop	e.g. *C:\Users\Public\Desktop*
$envCommonDocuments	e.g. *C:\Users\Public\Documents*
$envCommonStartMenuPrograms	e.g. *C:\ProgramData\Microsoft\Windows\Start Menu\Programs*
$envCommonStartMenu	e.g. *C:\ProgramData\Microsoft\Windows\Start Menu*
$envCommonStartUp	e.g. *C:\ProgramData\Microsoft\Windows\Start Menu*
$envCommonTemplates	e.g. *C:\ProgramData\Microsoft\Windows\Templates*
$envComputerName	$COMPUTERNAME% e.g. Computer1
$envComputerNameFQDN	Fully qualified computer name e.g. computer1.conto.contoso.com
$envHomeDrive	%HOMEDRIVE% e.g. *C:*
$envHomePath	%HOMEPATH% *C:\Users\%USERNAME%*

Variable	Description
$envHomeShare	%HOMESHARE% (Used instead of HOMEDRIVE if the home directory uses UNC paths.)
$envLocalAppData	%LOCALAPPDATA% e.g. *C:\Users\%USERNAME%\AppData\Local*
$envLogicalDrives	An array containing all of the logical drives on the system.
$envProgramFiles	%PROGRAMFILES% e.g. *C:\Program Files*
$envProgramFilesX86	%ProgramFiles(x86)% e.g. *C:\Program Files (x86)* (Only on 64 bit systems, is used to store 32 bit programs.)
$envProgramData	%PROGRAMDATA% e.g. *C:\ProgramData*
$envPublic	%PUBLIC% e.g. *C:\Users\Public*
$envSystemDrive	%SYSTEMDRIVE% e.g. *C:*
$envSystemRAM	System RAM as an integer
$envSystemRoot	%SYSTEMROOT% e.g. *C:\Windows*
$envTemp	Checks for the existence of environment variables in the following order and uses the first path found: - The path specified by the TMP environment variable. - The path specified by the TEMP (e.g. *C:\Users\%USERNAME%\AppData\Local\Temp*) environment variable. - The path specified by the USERPROFILE environment variable. - The Windows root (*C:\Windows*) directory.
$envUserCookies	*C:\Users\%USERNAME%\AppData\Local\Microsoft\Windows\INetCookies*
$envUserDesktop	*C:\Users\%USERNAME%\Desktop*
$envUserFavorites	*C:\Users\%USERNAME%\Favorites*
$envUserInternetCache	*C:\Users\%USERNAME%\AppData\Local\Microsoft\Windows\INetCache*
$envUserInternetHistory	*C:\Users\%USERNAME%\AppData\Local\Microsoft\Windows\History*
$envUserMyDocuments	*C:\Users\%USERNAME%\Documents*
$envUserName	%USERNAME%
$envUserProfile	%USERPROFILE% e.g. *%SystemDrive%\Users\%USERNAME%*
$envUserSendTo	*C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\SendTo*
$envUserStartMenu	*C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start Menu*
$envUserStartMenuPrograms	*C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs*
$envUserStartUp	*C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup*
$envSystem32Directory	*C:\WINDOWS\system32*
$envWinDir	%WINDIR% e.g. *C:\Windows*
Domain Membership	
$IsMachinePartOfDomain	Is machine joined to a domain (e.g. $true/$false)
$envMachineWorkgroup	If machine not joined to domain, what is the WORKGROUP it belongs to?
$envMachineADDomain	Root AD domain name for machine (e.g. domain.contoso.com)
$envLogonServer	FQDN of %LOGONSERVER% used for authenticating logged in user
$MachineDomainController	FQDN of an AD domain controller used for authentication
$envMachineDNSDomain	Full Domain name for machine (e.g. <name>.conto.contoso.com)

Variable	Description
$envUserDNSDomain	%USERDNSDOMAIN%. Root AD domain name for user (e.g. <name>.<suffix>.contoso.com)
$envUserDomain	%USERDOMAIN% (e.g. <name>.<suffix>.CONTOSO.<tld>)
Operating System	
$envOS	Object that contains details about the operating system
$envOSName	Name of the operating system (e.g. Microsoft Windows 8.1 Pro)
$envOSServicePack	Latest service pack installed on the system (e.g. Service Pack 3)
$envOSVersion	Full version number of the OS (e.g. {major}.{minor}.{build}.{revision})
$envOSVersionMajor	Major portion of the OS version number (e.g. {major}.{minor}.{build}.{revision})
$envOSVersionMinor	Minor portion of the OS version number (e.g. {major}.{minor}.{build}.{revision})
$envOSVersionBuild	Build portion of the OS version number (e.g. {major}.{minor}.{build}.{revision})
$envOSVersionRevision	Revision portion of the OS version number (e.g. {major}.{minor}.{build}.{revision})
$envOSProductType	OS product type represented as an integer (e.g. 1/2/3)
$IsServerOS	Is server OS? (e.g. $true/$false)
$IsDomainControllerOS	Is domain controller OS? (e.g. $true/$false)
$IsWorkStationOS	Is workstation OS? (e.g. $true/$false)
$envOSProductTypeName	OS product type name (e.g. Server/Domain Controller/Workstation/Unknown)
$Is64Bit	Is this a 64-bit OS? (e.g. $true/$false)
$envOSArchitecture	Represents the OS architecture (e.g. 32-Bit/64-Bit)
Current Process Architecture	
$Is64BitProcess	Is the current process 64-bits? (e.g. $true/$false)
$psArchitecture	Represents the current process architecture (e.g. x86/x64)
PowerShell And CLR (.NET) Versions	
$envPSVersionTable	Object containing PowerShell version details from PS variable $PSVersionTable
$envPSVersion	Full version number of PS (e.g. {major}.{minor}.{build}.{revision})
$envPSVersionMajor	Major portion of PS version number (e.g. {major}.{minor}.{build}.{revision})
$envPSVersionMinor	Minor portion of PS version number (e.g. {major}.{minor}.{build}.{revision})
$envPSVersionBuild	Build portion of PS version number (e.g. {major}.{minor}.{build}.{revision})
$envPSVersionRevision	Revision portion of PS version number (e.g. {major}.{minor}.{build}.{revision})
$envCLRVersion	Full version number of .NET used by PS (e.g. {major}.{minor}.{build}.{revision})
$envCLRVersionMajor	Major portion of PS .NET version number (e.g. {major}.{minor}.{build}.{revision})
$envCLRVersionMinor	Minor portion of PS .NET version number (e.g. {major}.{minor}.{build}.{revision})
$envCLRVersionBuild	Build portion of PS .NET version number (e.g. {major}.{minor}.{build}.{revision})
$envCLRVersionRevision	Revision portion of PS .NET version number (e.g. {major}.{minor}.{build}.{revision})
Permissions/Accounts	
$CurrentProcessToken	Object that represents the current processes Windows Identity user token. Contains all details regarding user permissions.
$CurrentProcessSID	Object that represents the current process account SID (e.g. S-1-5-32-544)
$ProcessNTAccount	Current process NT Account (e.g. NT AUTHORITY\SYSTEM)
$ProcessNTAccountSID	Current process account SID (e.g. S-1-5-32-544)

Variable	Description
$IsAdmin	Is the current process running with elevated admin privileges? (e.g. $true/$false)
$IsLocalSystemAccount	Is the current process running under the SYSTEM account? (e.g. $true/$false)
$IsLocalServiceAccount	Is the current process running under LOCAL SERVICE account? (e.g. $true/$false)
$IsNetworkServiceAccount	Is the current process running under the NETWORK SERVICE account? (e.g. $true/$false)
$IsServiceAccount	Is the current process running as a service? (e.g. $true/$false)
$IsProcessUserInteractive	Is the current process able to display a user interface?
$LocalSystemNTAccount	Localized NT account name of the SYSTEM account (e.g. NT AUTHORITY\SYSTEM)
$SessionZero	Is the current process currently in session zero? In session zero isolation, process is not able to display a user interface. (e.g. $true/$false)
Script Name and Script Paths	
$scriptPath	Fully qualified path of the toolkit e.g. *C:\Testing\AppDeployToolkit\AppDeployToolkitMain.ps1*
$scriptName	Name of toolkit without file extension e.g. AppDeployToolkitMain
$scriptFileName	Name of toolkit file e.g. *AppDeployToolkitMain.ps1*
$scriptRoot	Folder that the toolkit is located in. e.g. *C:\Testing\AppDeployToolkit*
$invokingScript	Fully qualified path of the script that invoked the toolkit e.g. *C:\Testing\Deploy-Application.ps1*
$scriptParentPath	If toolkit was invoked by another script: contains folder that the invoking script is located in.
	If toolkit was not invoked by another script: contains parent folder of the toolkit.
App Deploy Script Dependency Files	
$appDeployLogoIcon	Path to the logo icon file for the toolkit e.g. *$scriptRoot\AppDeployToolkitLogo.ico*
$appDeployLogoBanner	Path to the logo banner file for the toolkit e.g. *$scriptRoot\AppDeployToolkitBanner.png*
$appDeployConfigFile	Path to the config XML file for the toolkit e.g. *$scriptRoot\AppDeployToolkitConfig.xml*
$appDeployToolkitDotSourceExtensions	Name of the optional extensions file for the toolkit e.g. *AppDeployToolkitExtensions.ps1*
$xmlConfigFile	Contains the entire contents of the XML config file
$configConfigVersion	Version number of the config XML file
$configConfigDate	Last modified date of the config XML file
Script Directories	
$dirFiles	*Files* sub-directory of the toolkit
$dirSupportFiles	*SupportFiles* sub-directory of the toolkit
$dirAppDeployTemp	Toolkit temp directory. Configured in XML Config file option Toolkit_TempPath. e.g. *Toolkit_TempPath$appDeployToolkitName*
Script Naming Convention	
$appVendor	Name of the manufacturer that created the package being deployed (e.g. Microsoft)
$appName	Name of the application being packaged (e.g. Office 2010)
$appVersion	Version number of the application being packaged (e.g. 14.0)
$appLang	UI language of the application being packaged (e.g. EN)

Variable	Description
$appRevision	Revision number of the package (e.g. 01)
$appArch	Architecture of the application being packaged (e.g. x86/x64)
$installTitle	Combination of the most important details about the application being packaged (e.g. "$appVendor $appName $appVersion")
$installName	Combination of any of the following details which were provided:
	$appVendor + _ + $appName + _ + $appVersion + _ + $appArch + _ + $appLang + _ + $appRevision
Executables	
$exeWusa	Name of system utility that installs Standalone Windows Updates e.g. *wusa.exe*
$exeMsiexec	Name of system utility that install Windows Installer files e.g. *msiexec.exe*
$exeSchTasks	Path of system utility that allows management of scheduled tasks e.g. *$envWinDir\System32\schtasks.exe*
RegEx Patterns	
$MSIProductCodeRegExPattern	Contains the regex pattern used to detect a MSI product code.
Registry Keys	
$regKeyApplications	Array containing the path to the 32-bit and 64-bit portions of the registry that contain information about programs installed on the system. *HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall*
$regKeyLotusNotes	Contains the registry path that stores information about a Lotus Notes installation. *HKLM:SOFTWARE\Lotus\Notes* *HKLM:SOFTWARE\Wow6432Node\Lotus\Notes*
$regKeyAppExecution	Contains the registry path where application execution can be blocked by configuring the 'Debugger' value.
	HKLM:SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
$regKeyDeferHistory	The path in the registry where the defer history for the package being installed is stored.
	$configToolkitRegPath + $appDeployToolkitName\DeferHistory$installName
COM Objects	
$Shell	Represents and allows use of the WScript.Shell COM object
$ShellApp	Represents and allows use of the Shell.Application COM object
Log File	
$logName	Name of the script log file: $installName + '' + *$appDeployToolkitName* + " + $deploymentType + '.log'
$logTempFolder	Temporary log file directory used if the option to compress log files was selected in the config XML file: $envTemp$installName
$configToolkitLogDir	Path to log directory defined in XML config file
$DisableScriptLogging	Dot source this ScriptBlock to disable logging messages to the log file.
$RevertScriptLogging	Dot source this ScriptBlock to revert script logging back to its original setting.
Script Parameters	
$deployAppScriptParameters	Non-default parameters that Deploy-Application.ps1 was launched with
$appDeployMainScriptParameters	Non-default parameters that AppDeployToolkitMain.ps1 was launched with
$appDeployExtScriptParameters	Non-default parameters that AppDeployToolkitExtensions.ps1 was launched with

Variable	Description
Logged On Users	
$LoggedOnUserSessions	Object that contains account and session details for all users
$usersLoggedOn	Array that contains all of the NTAccount names of logged in users
$CurrentLoggedOnUserSession	Object that contains account and session details for the current process if it is running as a logged in user.
	This is the object from $LoggedOnUserSessions where the IsCurrentSession property is $true.
$CurrentConsoleUserSession	Objects that contains the account and session details of the console user (user with control of the physical monitor, keyboard, and mouse).
	This is the object from $LoggedOnUserSessions where the IsConsoleSession property is $true.
$RunAsActiveUser	The active console user. If no console user exists but users are logged in, such as on terminal servers, then the first logged-in non-console user.
Miscellaneous	
$dpiPixels	DPI Scale (property only exists if DPI scaling has been changed on the system at least once)
$runningTaskSequence	Is the current process running in a SCCM task sequence? (e.g. $true/$false)
$IsTaskSchedulerHealthy	Are the task scheduler services in a healthy state? (e.g. $true/$false)
$invalidFileNameChars	Array of all invalid file name characters used to sanitize variables which may be used to create file names.
$useDefaultMsi	A Zero-Config MSI installation was detected.

Reference - Toolkit Functions

Convert-RegistryPath

SYNOPSIS

Converts the specified registry key path to a format that is compatible with built-in PowerShell cmdlets.

SYNTAX

```
Convert-RegistryPath [-Key] <String> [[-SID] <String>] [<CommonParameters>]
```

DESCRIPTION

Converts the specified registry key path to a format that is compatible with built-in PowerShell cmdlets.

Converts registry key hives to their full paths. Example: HKLM is converted to "Registry::HKEY_LOCAL_MACHINE".

PARAMETERS

```
-Key <String>
```

Path to the registry key to convert (can be a registry hive or fully qualified path)

```
-SID <String>
```

The security identifier (SID) for a user. Specifying this parameter will convert a HKEY_CURRENT_USER registry key to the HKEY_USERS$SID format.

Specify this parameter from the `Invoke-HKCURegistrySettingsForAllUsers` function to read/edit HKCU registry settings for all users on the system.

EXAMPLE 1

```
Convert-RegistryPath -Key
'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\
{1AD147D0-BE0E-3D6C-AC11-64F6DC4163F1}'
```

EXAMPLE 2

```
Convert-RegistryPath -Key
'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{1AD147D0-BE0E-3D6C-
AC11-64F6DC4163F1}'
```

Copy-File

SYNOPSIS

Copy a file or group of files to a destination path.

SYNTAX

```
Copy-File [-Path] <String[]> [-Destination] <String> [-Recurse] [[-
ContinueOnError] <Boolean>] [[-ContinueFileCopyOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Copy a file or group of files to a destination path.

PARAMETERS

```
-Path <String[]>
```

Path of the file to copy.

```
-Destination <String>
```

Destination Path of the file to copy.

```
-Recurse [<SwitchParameter>]
```

Copy files in subdirectories.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. This will continue the deployment script, but will not continue copying files if an error is encountered. Default is: $true.

```
-ContinueFileCopyOnError <Boolean>
```

Continue copying files if an error is encountered. This will continue the deployment script and will warn about files that failed to be copied. Default is: $false.

EXAMPLE 1

```
Copy-File -Path "$dirSupportFiles\MyApp.ini" -Destination "$envWindir\MyApp.ini"
```

EXAMPLE 2

```
Copy-File -Path "$dirSupportFiles\*.*" -Destination "$envTemp\tempfiles"
```

Copy all of the files in a folder to a destination folder.

Disable-TerminalServerInstallMode

SYNOPSIS

Changes to user install mode for Remote Desktop Session Host/Citrix servers.

SYNTAX

```
Disable-TerminalServerInstallMode [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Changes to user install mode for Remote Desktop Session Host/Citrix servers.

PARAMETERS

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
 Disable-TerminalServerInstallMode
```

Enable-TerminalServerInstallMode

SYNOPSIS

Changes to user install mode for Remote Desktop Session Host/Citrix servers.

SYNTAX

```
Enable-TerminalServerInstallMode [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Changes to user install mode for Remote Desktop Session Host/Citrix servers.

PARAMETERS

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Enable-TerminalServerInstall
```

Execute-MSI

SYNOPSIS

Executes msiexec.exe to perform the following actions for MSI & MSP files and MSI product codes: install, uninstall, patch, repair, active setup.

If the -Action parameter is set to "Install" and the MSI is already installed, the function will exit.

Sets default switches to be passed to msiexec based on the preferences in the XML configuration file.

Automatically generates a log file name and creates a verbose log file for all msiexec operations.

Expects the MSI or MSP file to be located in the "Files" sub directory of the App Deploy Toolkit. Expects transform files to be in the same directory as the MSI file.

DESCRIPTION

Changes to user install mode for Remote Desktop Session Host/Citrix servers.

SYNTAX

```
Execute-MSI [[-Action] <String>] [-Path] <String> [[-Transform] <String>] [[-
Parameters] <String>] [[-AddParameters] <String>] [-SecureParameters] [[-Patch]
<String>] [[-LoggingOptions] <String>] [[-private:LogName] <String>] [[-
WorkingDirectory] <String>] [-SkipMSIAlreadyInstalledCheck] [-
IncludeUpdatesAndHotfixes] [-PassThru] [[-ContinueOnError] <Boolean>]
```

PARAMETERS

```
-Action <String>
```

The action to perform. Options: Install, Uninstall, Patch, Repair, ActiveSetup.

```
-Path <String>
```

The path to the MSI/MSP file or the product code of the installed MSI.

```
-Transform <String>
```

The name of the transform file(s) to be applied to the MSI. The transform file is expected to be in the same directory as the MSI file. Multiple transforms should be separated by a semi-colon.

```
-Parameters <String>
```

Overrides the default parameters specified in the XML configuration file. Install default is: "REBOOT=ReallySuppress /QB!". Uninstall default is: "REBOOT=ReallySuppress /QN".

```
-AddParameters <String>
```

Adds to the default parameters specified in the XML configuration file. Install default is: "REBOOT=ReallySuppress /QB!". Uninstall default is: "REBOOT=ReallySuppress /QN".

```
-SecureParameters [<SwitchParameter>]
```

Hides all parameters passed to the MSI or MSP file from the toolkit Log file.

```
-Patch <String>
```

The name of the patch (msp) file(s) to be applied to the MSI for use with the "Install" action. The patch file is expected to be in the same directory as the MSI file. Multiple transforms should be be separated by a semi-colon.

```
-LoggingOptions <String>
```

Overrides the default logging options specified in the XML configuration file. Default options are: "/L*v".

```
-private:LogName <String>
```

Overrides the log file name.

```
-WorkingDirectory <String>
```

Overrides the working directory. The working directory is set to the location of the MSI file.

```
-SkipMSIAlreadyInstalledCheck [<SwitchParameter>]
```

Skips the check to determine if the MSI is already installed on the system. Default is: $false.

```
-IncludeUpdatesAndHotfixes [<SwitchParameter>]
```

Include matches against updates and hotfixes in results.

```
-PassThru [<SwitchParameter>]
```

Returns ExitCode, STDOut, and STDErr output from the process.

```
-ContinueOnError <Boolean>
```

Continue if an exit code is returned by msiexec that is not recognized by the App Deploy Toolkit. Default is: $false.

EXAMPLE 1

```
Execute-MSI -Action 'Install' -Path 'Adobe_FlashPlayer_11.2.202.233_x64_EN.msi'
```

Installs an MSI

EXAMPLE 2

```
Execute-MSI -Action 'Install' -Path 'Adobe_FlashPlayer_11.2.202.233_x64_EN.msi'
-Transform 'Adobe_FlashPlayer_11.2.202.233_x64_EN_01.mst' -Parameters '/QN'
```

Installs an MSI, applying a transform and overriding the default MSI toolkit parameters

EXAMPLE 3

```
[psobject]$ExecuteMSIResult = Execute-MSI -Action 'Install' -Path
'Adobe_FlashPlayer_11.2.202.233_x64_EN.msi' -PassThru
```

Installs an MSI and stores the result of the execution into a variable by using the -PassThru option

EXAMPLE 4

```
Execute-MSI -Action 'Uninstall' -Path '{26923b43-4d38-484f-9b9e-de460746276c}'
```

Uninstalls an MSI using a product code

EXAMPLE 5

```
Execute-MSI -Action 'Patch' -Path 'Adobe_Reader_11.0.3_EN.msp'
```

Execute-MSP

SYNOPSIS

Reads SummaryInfo targeted product codes in MSP file and determines if the MSP file applies to any installed products

If a valid installed product is found, triggers the Execute-MSI function to patch the installation.

Uses default config MSI parameters. You can use -AddParameters to add additional parameters.

SYNTAX

```
Execute-MSP [-Path] <String> [<CommonParameters>]
```

PARAMETERS

```
-Path <String>
```

Path to the MSP file.

PARAMETERS

```
-AddParameters <String>
```

Additional Parameters.

EXAMPLE 1

```
Execute-MSP -Path 'Adobe_Reader_11.0.3_EN.msp'
```

EXAMPLE 2

```
Execute-MSP -Path 'AcroRdr2017Upd1701130143_MUI.msp' -AddParameters 'ALLUSERS=1'
```

Execute-Process

SYNOPSIS

Execute a process with optional arguments, working directory, window style.

SYNTAX

```
Execute-Process [-Path] <String> [[-Parameters] <String[]>] [-SecureParameters]
[[-WindowStyle] {Normal | Hidden | Minimized | Maximized}] [-CreateNoWindow] [[-
WorkingDirectory] <String>] [-NoWait] [-PassThru] [-WaitForMsiExec] [[-
MsiExecWaitTime] <TimeSpan>] [[-IgnoreExitCodes] <String>] [[-ContinueOnError]
<Boolean>] [<CommonParameters>]
```

DESCRIPTION

Executes a process, e.g. a file included in the Files directory of the App Deploy Toolkit, or a file on the local machine.

Provides various options for handling the return codes (see PARAMETERS).

PARAMETERS

```
-Path <String>
```

Path to the file to be executed. If the file is located directly in the "Files" directory of the App Deploy Toolkit, only the file name needs to be specified.

Otherwise, the full path of the file must be specified. If the files is in a subdirectory of "Files", use the "$dirFiles" variable as shown in the example.

```
-Parameters <String[]>
```

Arguments to be passed to the executable

```
-SecureParameters [<SwitchParameter>]
```

Hides all parameters passed to the executable from the Toolkit log file

```
-WindowStyle
```

Style of the window of the process executed. Options: Normal, Hidden, Maximized, Minimized. Default: Normal.
Note: Not all processes honor the "Hidden" flag. If it it not working, then check the command line options for the process being executed to see it has a silent option.

```
-CreateNoWindow [<SwitchParameter>]
```

Specifies whether the process should be started with a new window to contain it. Default is false.

```
-WorkingDirectory <String>
```

The working directory used for executing the process. Defaults to the directory of the file being executed.

```
-NoWait [<SwitchParameter>]
```

Immediately continue after executing the process.

```
-PassThru [<SwitchParameter>]
```

Returns ExitCode, STDOut, and STDErr output from the process.

```
-WaitForMsiExec [<SwitchParameter>]
```

Sometimes an EXE bootstrapper will launch an MSI install. In such cases, this variable will ensure that this function waits for the msiexec engine to become available before starting the install.

```
-MsiExecWaitTime <TimeSpan>
```

Specify the length of time in seconds to wait for the msiexec engine to become available. Default: 600 seconds (10 minutes).

```
-IgnoreExitCodes <String>
```

List the exit codes to ignore.

```
-ContinueOnError <Boolean>
```

Continue if an exit code is returned by the process that is not recognized by the App Deploy Toolkit. Default: $false.

EXAMPLE 1

```
Execute-Process -Path 'uninstall_flash_player_64bit.exe' -Parameters
'/uninstall' -WindowStyle 'Hidden'
```

If the file is in the "Files" directory of the App Deploy Toolkit, only the file name needs to be specified.

EXAMPLE 2

```
Execute-Process -Path "$dirFiles\Bin\setup.exe" -Parameters '/S' -WindowStyle
'Hidden'
```

EXAMPLE 3

```
Execute-Process -Path 'setup.exe' -Parameters '/S' -IgnoreExitCodes '1,2'
```

EXAMPLE 4

```
Execute-Process -Path 'setup.exe' -Parameters "-s -
f2`"$configToolkitLogDir\$installName.log`""
```

Launch InstallShield "setup.exe" from the ".\Files" sub-directory and force log files to the logging folder.

EXAMPLE 5

```
Execute-Process -Path 'setup.exe' -Parameters "/s /v`"ALLUSERS=1 /qn /L\*
\`"$configToolkitLogDir\$installName.log`"`""
```

Launch InstallShield "setup.exe" with embedded MSI and force log files to the logging folder.

Execute-ProcessAsUser

SYNOPSIS

Execute a process with a logged in user account, by using a scheduled task, to provide interaction with user in the SYSTEM context.

SYNTAX

```
Execute-ProcessAsUser [[-UserName] <String>] [-Path] <String> [[-Parameters]
<String>] [-SecureParameters] [[-RunLevel] <String>] [-Wait] [-PassThru] [[-
ContinueOnError] <Boolean>]
```

DESCRIPTION

Execute a process with a logged in user account, by using a scheduled task, to provide interaction with user in the SYSTEM context.

PARAMETERS

```
-UserName <String>
```

Logged in Username under which to run the process from. Default is: The active console user. If no console user exists but users are logged in, such as on terminal servers, then the first logged-in non-console user.

```
-Path <String>
```

Path to the file being executed.

```
-Parameters <String>
```

Arguments to be passed to the file being executed.

```
-SecureParameters [<SwitchParameter>]
```

Hides all parameters passed to the executable from the Toolkit log file.

```
-RunLevel <String>
```

Specifies the level of user rights that Task Scheduler uses to run the task. The acceptable values for this parameter are:

- HighestAvailable: Tasks run by using the highest available privileges (Admin privileges for Administrators). Default Value.
- LeastPrivilege: Tasks run by using the least-privileged user account (LUA) privileges.

```
-Wait [<SwitchParameter>]
```

Wait for the process, launched by the scheduled task, to complete execution before accepting more input. Default is $false.

```
-PassThru [<SwitchParameter>]
```

Returns the exit code from this function or the process launched by the scheduled task.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is $true.

EXAMPLE 1

```
Execute-ProcessAsUser -UserName 'CONTOSO\User' -Path "$PSHOME\powershell.exe" -
Parameters "-Command & { & `"C:\Test\Script.ps1`"; Exit `$LastExitCode }" -Wait
```

Execute process under a user account by specifying a username under which to execute it.

EXAMPLE 2

```
Execute-ProcessAsUser -Path "$PSHOME\powershell.exe" -Parameters "-Command & { &
`"C:\Test\Script.ps1`"; Exit `$LastExitCode }" -Wait
```

Execute process under a user account by using the default active logged in user that was detected when the toolkit was launched.

Exit-Script

SYNOPSIS

Exit the script, perform cleanup actions, and pass an exit code to the parent process.

SYNTAX

```
Exit-Script [[-ExitCode] <Int32>] [<CommonParameters>]
```

DESCRIPTION

Always use when exiting the script to ensure cleanup actions are performed.

PARAMETERS

```
-ExitCode <Int32>
```

The exit code to be passed from the script to the parent process, e.g. SCCM

EXAMPLE 1

```
Exit-Script
```

EXAMPLE 2

```
Exit-Script -ExitCode 1618
```

Get-FileVersion

SYNOPSIS

Gets the version of the specified file

SYNTAX

```
Get-FileVersion [-File] <String> [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Gets the version of the specified file

PARAMETERS

```
-File <String>
```

Path of the file

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Get-FileVersion -File "$envProgramFilesX86\Adobe\Reader
11.0\Reader\AcroRd32.exe"
```

Get-FreeDiskSpace

SYNOPSIS

Retrieves the free disk space in MB on a particular drive (defaults to system drive)

SYNTAX

```
Get-FreeDiskSpace [[-Drive] <String>] [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Retrieves the free disk space in MB on a particular drive (defaults to system drive)

PARAMETERS

```
-Drive <String>
```

Drive to check free disk space on

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Get-FreeDiskSpace -Drive 'C:'
```

Get-HardwarePlatform

SYNOPSIS

Retrieves information about the hardware platform (physical or virtual)

SYNTAX

```
Get-HardwarePlatform [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Retrieves information about the hardware platform (physical or virtual)

PARAMETERS

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Get-HardwarePlatform
```

Get-HardwarePlatform

SYNOPSIS

Retrieves information about the hardware platform (physical or virtual)

SYNTAX

```
Get-HardwarePlatform [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Retrieves information about the hardware platform (physical or virtual)

PARAMETERS

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Get-HardwarePlatform
```

Get-IniValue

SYNOPSIS

Parses an INI file and returns the value of the specified section and key.

SYNTAX

```
Get-IniValue [-FilePath] <String> [-Section] <String> [-Key] <String> [[-
ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Parses an INI file and returns the value of the specified section and key.

PARAMETERS

```
-FilePath <String>
```

Path to the INI file.

```
-Section <String>
```

Section within the INI file.

```
-Key <String>
```

Key within the section of the INI file.

```
-ContinueOnError <Boolean>
```

EXAMPLE 1

```
Get-IniValue -FilePath "$envProgramFilesX86\IBM\Notes\notes.ini" -Section
'Notes' -Key 'KeyFileName'
```

Get-InstalledApplication

SYNOPSIS

Retrieves information about installed applications.

SYNTAX

```
Get-InstalledApplication [[-Name] <String[]>] [-Exact] [-WildCard] [-RegEx] [[-
ProductCode] <String>] [-IncludeUpdatesAndHotfixes] [<CommonParameters>]
```

DESCRIPTION

Retrieves information about installed applications by querying the registry. You can specify an application name, a product code, or both.

Returns information about application publisher, name & version, product code, uninstall string, install source, location, date, and application architecture.

PARAMETERS

```
-Name <String[]>
```

The name of the application to retrieve information for. Performs a contains match on the application display name by default.

```
-Exact [<SwitchParameter>]
```

Specifies that the named application must be matched using the exact name.

```
-WildCard [<SwitchParameter>]
```

Specifies that the named application must be matched using a wildcard search.

```
-RegEx [<SwitchParameter>]
```

Specifies that the named application must be matched using a regular expression search.

```
-ProductCode <String>
```

The product code of the application to retrieve information for.

```
-IncludeUpdatesAndHotfixes [<SwitchParameter>]
```

Include matches against updates and hotfixes in results.

EXAMPLE 1

```
Get-InstalledApplication -Name 'Adobe Flash'
```

EXAMPLE 2

```
Get-InstalledApplication -ProductCode '{1AD147D0-BE0E-3D6C-AC11-64F6DC4163F1}'
```

Get-LoggedOnUser

SYNOPSIS

Get session details for all local and RDP logged on users.

SYNTAX

```
Get-LoggedOnUser [<CommonParameters>]
```

DESCRIPTION

Get session details for all local and RDP logged on users using Win32 APIs. Get the following session details:
NTAccount, SID, UserName, DomainName, SessionId, SessionName, ConnectState, IsCurrentSession, IsConsoleSession, IsUserSession, IsActiveUserSession IsRdpSession, IsLocalAdmin, LogonTime, IdleTime, DisconnectTime, ClientName, ClientProtocolType, ClientDirectory, ClientBuildNumber

EXAMPLE 1

```
Get-LoggedOnUser
```

NOTES
Description of ConnectState property:

Value	Description
Active	A user is logged on to the session.
ConnectQuery	The session is in the process of connecting to a client.
Connected	A client is connected to the session.
Disconnected	The session is active, but the client has disconnected from it.
Down	The session is down due to an error.
Idle	The session is waiting for a client to connect.
Initializing	The session is initializing.
Listening	The session is liste ning for connections.
Reset	The session is being reset.
Shadowing	This session is shadowing another session.

Description of IsActiveUserSession property:

- If a console user exists, then that will be the active user session.
- If no console user exists but users are logged in, such as on terminal servers, then the first logged-in non-console user that has ConnectState either 'Active' or 'Connected' is the active user.

Description of IsRdpSession property:

- Gets a value indicating whether the user is associated with an RDP client session.

Description of IsLocalAdmin property:

- Checks whether the user is a member of the Administrators group

---

Get-PendingReboot

SYNOPSIS

Get the pending reboot status on a local computer.

SYNTAX

```
Get-PendingReboot [<CommonParameters>]
```

DESCRIPTION

Check WMI and the registry to determine if the system has a pending reboot operation from any of the following:
a) Component Based Servicing (Vista, Windows 2008)
b) Windows Update / Auto Update (XP, Windows 2003 / 2008)
c) SCCM 2012 Clients (DetermineIfRebootPending WMI method)
d) App-V Pending Tasks (global based Appv 5.0 SP2)
e) Pending File Rename Operations (XP, Windows 2003 / 2008)

EXAMPLE 1

```
Get-PendingReboot
```

EXAMPLE 2

```
(Get-PendingReboot).IsSystemRebootPending
```

Returns boolean value determining whether or not there is a pending reboot operation.

NOTES

Returns custom object with following properties:
ComputerName, LastBootUpTime, IsSystemRebootPending, IsCBServicingRebootPending, IsWindowsUpdateRebootPending, IsSCCMClientRebootPending, IsFileRenameRebootPending, PendingFileRenameOperations, ErrorMsg

ErrorMsg only contains something if an error occurred.

Get-RegistryKey

SYNOPSIS

Retrieves value names and value data for a specified registry key or optionally, a specific value.

SYNTAX

```
Get-RegistryKey [-Key] <String> [[-Value] <String>] [[-SID] <String>] [-
ReturnEmptyKeyIfExists] [-DoNotExpandEnvironmentNames] [[-ContinueOnError]
<Boolean>] [<CommonParameters>]
```

DESCRIPTION

Retrieves value names and value data for a specified registry key or optionally, a specific value.

If the registry key does not exist or contain any values, the function will return $null by default. To test for existence of a registry key path, use built-in Test-Path cmdlet.

PARAMETERS

```
-Key <String>
```

Path of the registry key.

```
-Value <String>
```

Value to retrieve (optional).

```
-SID <String>
```

The security identifier (SID) for a user. Specifying this parameter will convert a HKEY_CURRENT_USER registry key to the HKEY_USERS$SID format.
Specify this parameter from the Invoke-HKCURegistrySettingsForAllUsers function to read/edit HKCU registry settings for all users on the system.

```
-ReturnEmptyKeyIfExists [<SwitchParameter>]
```

Return the registry key if it exists but it has no property/value pairs underneath it. Default is: $false.

```
-DoNotExpandEnvironmentNames [<SwitchParameter>]
```

Return unexpanded REG_EXPAND_SZ values. Default is: $false.

```
-ContinueOnError <Boolean>
```

EXAMPLE 1

```
Get-RegistryKey -Key 'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\
{1AD147D0-BE0E-3D6C-AC11-64F6DC4163F1}'
```

EXAMPLE 2

```
Get-RegistryKey -Key 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Image File Execution Options\iexplore.exe'
```

EXAMPLE 3

```
Get-RegistryKey -Key 'HKLM:Software\Wow6432Node\Microsoft\Microsoft SQL Server
Compact Edition\v3.5' -Value 'Version'
```

EXAMPLE 4

```
Get-RegistryKey -Key
'HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment'
 -Value 'Path' -DoNotExpandEnvironmentNames
```

Returns %ProgramFiles%\Java instead of C:\Program Files\Java

EXAMPLE 5

```
Get-RegistryKey -Key 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Example' -Value
'(Default)'
```

Get-SchedulerTask

SYNOPSIS

Retrieve all details for scheduled tasks on the local computer.

SYNTAX

```
Get-SchedulerTask [[-TaskName] <String>] [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Retrieve all details for scheduled tasks on the local computer using schtasks.exe. All property names have spaces and colons removed.

PARAMETERS

```
-TaskName <String>
```

Specify the name of the scheduled task to retrieve details for. Uses regex match to find scheduled task.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default: $true.

EXAMPLE 1

```
Get-SchedulerTask
```

To display a list of all scheduled task properties.

EXAMPLE 2

```
Get-SchedulerTask | Out-GridView
```

To display a grid view of all scheduled task properties.

EXAMPLE 3

```
Get-SchedulerTask | Select-Object -Property TaskName
```

To display a list of all scheduled task names.

NOTES

This function has an alias: Get-ScheduledTask if Get-ScheduledTask is not defined

---

Get-ServiceStartMode

SYNOPSIS

Get the service startup mode.

SYNTAX

```
Get-ServiceStartMode [-Name] <String> [[-ComputerName] <String>] [[-
ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Get the service startup mode.

PARAMETERS

```
-Name <String>
```

Specify the name of the service.

```
-ComputerName <String>
```

Specify the name of the computer. Default is: the local computer.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Get-ServiceStartMode -Name 'wuauserv'
```

Get-Shortcut

SYNOPSIS

Get information from a new .lnk or .url type shortcut.

SYNTAX

```
Get-Shortcut [-Path] <String> [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Get information from a new .lnk or .url type shortcut. Returns a hashtable.

PARAMETERS

```
-Path <String>
```

Path to the shortcut to get information from.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Get-Shortcut -Path "$envProgramData\Microsoft\Windows\Start Menu\My
Shortcut.lnk"
```

NOTES

Url shortcuts only support TargetPath, IconLocation and IconIndex.

Get-UniversalDate

SYNOPSIS

Returns the date/time for the local culture in a universal sortable date time pattern.

SYNTAX

```
Get-UniversalDate [[-DateTime] <String>] [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Converts the current datetime or a datetime string for the current culture into a universal sortable date time pattern, e.g. 2013-08-22 11:51:52Z

PARAMETERS

```
-DateTime <String>
```

Specify the DateTime in the current culture.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default: $false.

EXAMPLE 1

```
Get-UniversalDate
```

Returns the current date in a universal sortable date time pattern.

EXAMPLE 2

```
Get-UniversalDate -DateTime '25/08/2013'
```

Returns the date for the current culture in a universal sortable date time pattern.

Get-UserProfiles

SYNOPSIS

Get the User Profile Path, User Account Sid, and the User Account Name for all users that log onto the machine and also the Default User (which does not log on).

SYNTAX

```
Get-UserProfiles [[-ExcludeNTAccount] <String[]>] [[-ExcludeSystemProfiles]
<Boolean>] [-ExcludeDefaultUser] [<CommonParameters>]
```

DESCRIPTION

Get the User Profile Path, User Account Sid, and the User Account Name for all users that log onto the machine and also the Default User (which does not log on).
Please note that the NTAccount property ma y be empty for some user profiles but the SID and ProfilePath properties will always be populated.

PARAMETERS

```
-ExcludeNTAccount <String[]>
```

Specify NT account names in Domain\Username format to exclude from the list of user profiles.

```
-ExcludeSystemProfiles <Boolean>
```

Exclude the Default User. Default is: $false.

EXAMPLE 1

```
Get-UserProfiles
```

Returns the following properties for each user profile on the system: NTAccount, SID, ProfilePath

EXAMPLE 3

```
[string[]]$ProfilePaths = Get-UserProfiles | Select-Object -ExpandProperty
'ProfilePath'
```

Returns and expands the profile paths on the system. This information can then be used to make modifications under the user profile on the filesystem.

Get-WindowTitle

SYNOPSIS

Search for a window title.

SYNTAX

```
Get-WindowTitle -GetAllWindowTitles [-DisableFunctionLogging]
[<CommonParameters>]
```

```
Get-WindowTitle -WindowTitle <String> [-DisableFunctionLogging]
[<CommonParameters>]
```

DESCRIPTION

Search for a window title. If window title searched for returns more than one result, then details for each window will be displayed.

PARAMETERS

```
-WindowTitle <String>
```

The title of the application window to search for using regex matching.

```
-GetAllWindowTitles [<SwitchParameter>]
```

Get titles for all open windows on the system.

```
-DisableFunctionLogging [<SwitchParameter>]
```

Disables logging messages to the script log file.

EXAMPLE 1

```
Get-WindowTitle -WindowTitle 'Microsoft Word'
```

Gets details for each window that has the words "Microsoft Word" in the title.

EXAMPLE 2

```
Get-WindowTitle -GetAllWindowTitles
```

Gets details for all windows with a title.

EXAMPLE 3

```
Get-WindowTitle -GetAllWindowTitles | Where-Object { $_.ParentProcess -eq
'WINWORD' }
```

Get details for all windows belonging to Microsoft Word process with name "WINWORD".

NOTES

Returns the following properties for each window:

- WindowTitle
- WindowHandle
- ParentProcess
- ParentProcessMainWindowHandle
- ParentProcessId

Function does not work in SYSTEM context unless launched with "psexec.exe -s -i" to run it as an interactive process under the SYSTEM account.

Install-MSUpdates

SYNOPSIS

Install all Microsoft Updates in a given directory.

SYNTAX

```
Install-MSUpdates [-Directory] <String> [<CommonParameters>]
```

DESCRIPTION

Install all Microsoft Updates of type ".exe", ".msu", or ".msp" in a given directory (recursively search directory).

PARAMETERS

```
-Directory <String>
```

Directory containing the updates.

EXAMPLE 1

```
Install-MSUpdates -Directory "$dirFiles\MSUpdates"
```

Install-SCCMSoftwareUpdates

SYNOPSIS

Scans for outstanding SCCM updates to be installed and installs the pending updates.

SYNTAX

```
Install-SCCMSoftwareUpdates [[-SoftwareUpdatesScanWaitInSeconds] <Int32>] [[-
WaitForPendingUpdatesTimeout] <TimeSpan>] [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Scans for outstanding SCCM updates to be installed and installs the pending updates.
Only compatible with SCCM 2012 Client or higher. This function can take several minutes to run.

PARAMETERS

```
-SoftwareUpdatesScanWaitInSeconds <Int32>
```

The amount of time to wait in seconds for the software updates scan to complete. Default is: 180 seconds.

```
-WaitForPendingUpdatesTimeout <TimeSpan>
```

The amount of time to wait for missing and pending updates to install before exiting the function. Default is: 45 minutes.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Install-SCCMSoftwareUpdates
```

Invoke-HKCURegistrySettingsForAllUsers

SYNOPSIS

Set current user registry settings for all current users and any new users in the future.

SYNTAX

```
Invoke-HKCURegistrySettingsForAllUsers [-RegistrySettings] <ScriptBlock> [[-
UserProfiles] <PSObject[]>] [<CommonParameters>]
```

DESCRIPTION

Set HKCU registry settings for all current and future users by loading their NTUSER.dat registry hive file, and making the modifications.

This function will modify HKCU settings for all users even when executed under the SYSTEM account.

To ensure new users in the future get the registry edits, the Default User registry hive used to provision the registry for new users is modified.

This function can be used as an alternative to using ActiveSetup for registry settings.
The advantage of using this function over ActiveSetup is that a user does not have to log off and log back on before the changes take effect.

PARAMETERS

```
-RegistrySettings <ScriptBlock>
```

Script block which contains HKCU registry settings which should be modified for all users on the system. Must specify the -SID parameter for all HKCU settings.

```
-UserProfiles <PSObject[]>
```

Specify the user profiles to modify HKCU registry settings for. Default is all user profiles except for system profiles.

EXAMPLE 1

```
[scriptblock]$HKCURegistrySettings = {
  Set-RegistryKey -Key 'HKCU\Software\Microsoft\Office\14.0\Common' -Name
'qmenable' -Value 0 -Type DWord -SID $UserProfile.SID
```

```
Set-RegistryKey -Key 'HKCU\Software\Microsoft\Office\14.0\Common' -Name
'updatereliabilitydata' -Value 1 -Type DWord -SID $UserProfile.SID
}
Invoke-HKCURegistrySettingsForAllUsers -RegistrySettings $HKCURegistrySettings
```

Invoke-RegisterOrUnregisterDLL

SYNOPSIS

Register or unregister a DLL file.

SYNTAX

```
Invoke-RegisterOrUnregisterDLL [-FilePath] <String> [[-DLLAction] <String>] [[-
ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Register or unregister a DLL file using regsvr32.exe. Function can be invoked using alias: 'Register-DLL' or 'Unregister-DLL'.

PARAMETERS

```
-FilePath <String>
```

Path to the DLL file.

```
-DLLAction <String>
```

Specify whether to register or unregister the DLL. Optional if function is invoked using 'Register-DLL' or 'Unregister-DLL' alias.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Register-DLL -FilePath "C:\Test\DcTLSFileToDMSComp.dll"
```

Register DLL file using the "Register-DLL" alias for this function

EXAMPLE 2

```
UnRegister-DLL -FilePath "C:\Test\DcTLSFileToDMSComp.dll"
```

Unregister DLL file using the "Unregister-DLL" alias for this function

EXAMPLE 3

```
Invoke-RegisterOrUnregisterDLL -FilePath "C:\Test\DcTLSFileToDMSComp.dll" -
DLLAction 'Register'
```

Register DLL file using the actual name of this function

Invoke-SCCMTask

SYNOPSIS

Triggers SCCM to invoke the requested schedule task id.

SYNTAX

```
Invoke-SCCMTask [-ScheduleID] <String> [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Triggers SCCM to invoke the requested schedule task id.

PARAMETERS

```
-ScheduleID <String>
```

Name of the schedule id to trigger.

Options:

- HardwareInventory, SoftwareInventory, HeartbeatDiscovery, SoftwareInventoryFileCollection, RequestMachinePolicy, EvaluateMachinePolicy, LocationServicesCleanup, SoftwareMeteringReport, SourceUpdate, PolicyAgentCleanup, RequestMachinePolicy2, CertificateMaintenance, PeerDistributionPointStatus, PeerDistributionPointProvisioning, ComplianceIntervalEnforcement, SoftwareUpdatesAgentAssignmentEvaluation, UploadStateMessage, StateMessageManager, SoftwareUpdatesScan, AMTProvisionCycle, UpdateStorePolicy, StateSystemBulkSend, ApplicationManagerPolicyAction, PowerManagementStartSummarizer

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Invoke-SCCMTask 'SoftwareUpdatesScan'
```

EXAMPLE 2

```
Invoke-SCCMTask
```

New-Folder

SYNOPSIS

Create a new folder.

SYNTAX

```
New-Folder [-Path] <String> [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Create a new folder if it does not exist.

PARAMETERS

```
-Path <String>
```

Path to the new folder to create.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
New-Folder -Path "$envWinDir\System32"
```

New-MsiTransform

SYNOPSIS

Create a transform file for an MSI database.

SYNTAX

```
New-MsiTransform [-MsiPath] <String> [[-ApplyTransformPath] <String>] [[-
NewTransformPath] <String>] [-TransformProperties] <Hashtable> [[-
ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Create a transform file for an MSI database and create/modify properties in the Properties table.

PARAMETERS

```
-MsiPath <String>
```

Specify the path to an MSI file.

```
-ApplyTransformPath <String>
```

Specify the path to a transform which should be applied to the MSI database before any new properties are created or modified.

```
-NewTransformPath <String>
```

Specify the path where the new transform file with the desired properties will be created. If a transform file of the same name already exists, it will be deleted before a new one is created.
Default is:
  a) If -ApplyTransformPath was specified but not -NewTransformPath, then *ApplyTransformPath.new.mst*
  b) If only -MsiPath was specified, then *MsiPath.mst*

```
-TransformProperties <Hashtable>
```

Hashtable which contains calls to Set-MsiProperty for configuring the desired properties which should be included in new transform file.
Example hashtable: `[hashtable] $TransformProperties = @{ 'ALLUSERS' = '1' }`

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
[hashtable]$TransformProperties = {
  'ALLUSERS' = '1'
  'AgreeToLicense' = 'Yes'
  'REBOOT' = 'ReallySuppress'
  'RebootYesNo' = 'No'
  'ROOTDRIVE' = 'C:'
}
New-MsiTransform -MsiPath 'C:\Temp\PSADTInstall.msi' -TransformProperties
$TransformProperties
```

New-Shortcut

SYNOPSIS

Creates a new .lnk or .url type shortcut

SYNTAX

```
New-Shortcut [-Path] <String> [-TargetPath] <String> [[-Arguments] <String>] [[-
IconLocation] <String>] [[-IconIndex] <String>] [[-Description] <String>] [[-
WorkingDirectory] <String>] [[-WindowStyle] <String>] [-RunAsAdmin] [[-Hotkey]
<String>] [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Creates a new shortcut .lnk or .url file, with configurable options

PARAMETERS

```
-Path <String>
```

Path to save the shortcut

```
-TargetPath <String>
```

Target path or URL that the shortcut launches

```
-Arguments <String>
```

Arguments to be passed to the target path

```
-IconLocation <String>
```

Location of the icon used for the shortcut

```
-IconIndex <String>
```

The index of the icon. Executables, DLLs, ICO files with multiple icons need the icon index to be specified. This parameter is an Integer. The first index is 0.

```
-Description <String>
```

Description of the shortcut

```
-WorkingDirectory <String>
```

Working Directory to be used for the target path

```
-WindowStyle <String>
```

Windows style of the application. Options: Normal, Maximized, Minimized. Default is: Normal.

```
-RunAsAdmin [<SwitchParameter>]
```

Set shortcut to run program as administrator. This option will prompt user to elevate when executing shortcut.

```
-Hotkey <String>
```

Create a Hotkey to launch the shortcut, e.g. "CTRL+SHIFT+F".

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
New-Shortcut -Path "$envProgramData\Microsoft\Windows\Start Menu\My
Shortcut.lnk" -TargetPath "$envWinDir\system32\notepad.exe" -IconLocation
"$envWinDir\system32\notepad.exe" -Description 'Notepad' -WorkingDirectory
"$envHomeDrive\$envHomePath"
```

NOTES

Url shortcuts only support TargetPath, IconLocation and IconIndex. Other parameters are ignored.

Remove-File

SYNOPSIS

Removes one or more items from a given path on the filesystem.

SYNTAX

```
Remove-File -Path <String[]> [-Recurse] [-ContinueOnError <Boolean>]
[<CommonParameters>]
```

```
Remove-File -LiteralPath <String[]> [-Recurse] [-ContinueOnError <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Removes one or more items from a given path on the filesystem.

PARAMETERS

```
-Path <String[]>
```

Specifies the path on the filesystem to be resolved. The value of Path will accept wildcards. Will accept an array of values.

```
-LiteralPath <String[]>
```

Specifies the path on the filesystem to be resolved. The value of LiteralPath is used exactly as it is typed; no characters are interpreted as wildcards. Will accept an array of values.

```
-Recurse [<SwitchParameter>]
```

Deletes the files in the specified location(s) and in all child items of the location(s).

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Remove-File -Path 'C:\Windows\Downloaded Program Files\Temp.inf'
```

EXAMPLE 2

```
Remove-File -LiteralPath 'C:\Windows\Downloaded Program Files' -Recurse
```

Remove-Folder

SYNOPSIS

Remove folder and files if they exist.

SYNTAX

```
Remove-Folder [-Path] <String> [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Remove folder and all files recursively in a given path.

PARAMETERS

```
-Path <String>
```

Path to the folder to remove.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Remove-Folder -Path "$envWinDir\Downloaded Program Files"
```

Remove-InvalidFileNameChars

SYNOPSIS

Remove invalid characters from the supplied string.

SYNTAX

```
Remove-InvalidFileNameChars[-Name] <String> [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Remove invalid characters from the supplied string and returns a valid filename as a string.

PARAMETERS

```
-Name <String>
```

Text to remove invalid filename characters from.

EXAMPLE 1

```
Remove-InvalidFileNameChars -Name "Filename/\1"
```

Remove-MSIApplications

SYNOPSIS

Removes all MSI applications matching the specified application name.

SYNTAX

```
Remove-MSIApplications [-Name] <String> [-Exact] [-WildCard] [[-Parameters]
<String>] [[-AddParameters] <String>] [[-FilterApplication] <Array>] [[-
ExcludeFromUninstall] <Array>] [-IncludeUpdatesAndHotfixes] [[-LoggingOptions]
<String>] [[-private:LogName] <String>] [-PassThru] [[-ContinueOnError]
<Boolean>] [<CommonParameters>]
```

DESCRIPTION

Removes all MSI applications matching the specified application name. Enumerates the registry for installed applications matching the specified application name and uninstalls that application using the product code, provided the uninstall string matches "msiexec".

PARAMETERS

```
-Name <String>
```

The name of the application to uninstall. Performs a contains match on the application display name by default.

```
-Exact [<SwitchParameter>]
```

Specifies that the named application must be matched using the exact name.

```
-WildCard [<SwitchParameter>]
```

Specifies that the named application must be matched using a wildcard search.

```
-Parameters <String>
```

Overrides the default parameters specified in the XML configuration file. Uninstall default is: "REBOOT=ReallySuppress /QN".

```
-AddParameters <String>
```

Adds to the default parameters specified in the XML configuration file. Uninstall default is: "REBOOT=ReallySuppress /QN".

```
-FilterApplication <Array>
```

Two-dimensional array that contains one or more (property, value, match-type) sets that should be used to filter the list of results returned by Get-InstalledApplication to only those that should be uninstalled.
Properties that can be filtered upon:

- ProductCode

- DisplayName
- DisplayVersion
- UninstallString
- InstallSource
- InstallLocation
- InstallDate
- Publisher
- Is64BitApplication

```
-ExcludeFromUninstall <Array>
```

Two-dimensional array that contains one or more (property, value, match-type) sets that should be excluded from uninstall if found.
Properties that can be excluded:

- ProductCode
- DisplayName
- DisplayVersion
- UninstallString
- InstallSource
- InstallLocation
- InstallDate
- Publisher
- Is64BitApplication

```
-IncludeUpdatesAndHotfixes [<SwitchParameter>]
```

Include matches against updates and hotfixes in results.

```
-LoggingOptions <String>
```

Overrides the default logging options specified in the XML configuration file. Default options are: "/L*v".

```
-private:LogName <String>
```

Overrides the default logfile name.

```
-PassThru [<SwitchParameter>]
```

Returns ExitCode, STDOut, and STDErr output from the process.

```
-ContinueOnError <Boolean>
```

Continue if an exit code is returned by msiexec that is not recognized by the App Deploy Toolkit. Default is: $true.

EXAMPLE 1

```
Remove-MSIApplications -Name 'Adobe Flash'
```

Removes all versions of software that match the name "Adobe Flash"

EXAMPLE 2

```
Remove-MSIApplications -Name 'Adobe'
```

Removes all versions of software that match the name "Adobe"

EXAMPLE 3

```
Remove-MSIApplications -Name 'Java 8 Update' -FilterApplication
('Is64BitApplication', $false, 'Exact'),('Publisher', 'Oracle Corporation',
'Exact')
```

Removes all versions of software that match the name "Java 8 Update" where the software is 32-bits and the publisher is "Oracle Corporation".

EXAMPLE 4

```
Remove-MSIApplications -Name 'Java 8 Update' -FilterApplication (,('Publisher',
'Oracle Corporation', 'Exact')) -ExcludeFromUninstall (,('DisplayName', 'Java 8
Update 45', 'Contains'))
```

Removes all versions of software that match the name "Java 8 Update" and also have "Oracle Corporation" as the Publisher; however, it does not uninstall "Java 8 Update 45" of the software.

NOTE: if only specifying a single row in the two-dimensional arrays, the array must have the extra parentheses and leading comma as in this example.

EXAMPLE 5

```
Remove-MSIApplications -Name 'Java 8 Update' -ExcludeFromUninstall (,
('DisplayName', 'Java 8 Update 45', 'Contains'))
```

Removes all versions of software that match the name "Java 8 Update"; however, it does not uninstall "Java 8 Update 45" of the software.

NOTE: if only specifying a single row in the two-dimensional array, the array must have the extra parentheses and leading comma as in this example.

EXAMPLE 6

```
Remove-MSIApplications -Name 'Java 8 Update' -ExcludeFromUninstall
  ('Is64BitApplication', $true, 'Exact'),
  ('DisplayName', 'Java 8 Update 45', 'Exact'),
  ('DisplayName', 'Java 8 Update 4*', 'WildCard'),
  ('DisplayName', 'Java \d Update \d{3}', 'RegEx'),
  ('DisplayName', 'Java 8 Update', 'Contains')
```

Removes all versions of software that match the name "Java 8 Update"; however, it does not uninstall 64-bit versions of the software, Update 45 of the software, or any Update that starts with 4.

Remove-RegistryKey

SYNOPSIS

Deletes the specified registry key or value.

SYNTAX

```
Remove-RegistryKey [-Key] <String> [[-Name] <String>] [-Recurse] [[-SID]
<String>] [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Deletes the specified registry key or value.

PARAMETERS

```
-Key <String>
```

Path of the registry key to delete.

```
-Name <String>
```

Name of the registry value to delete.

```
-Recurse [<SwitchParameter>]
```

Delete registry key recursively.

```
-SID <String>
```

The security identifier (SID) for a user. Specifying this parameter will convert a HKEY_CURRENT_USER registry key to the HKEY_USERS$SID format.
Specify this parameter from the Invoke-HKCURegistrySettingsForAllUsers function to read/edit HKCU registry settings for all users on the system.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Remove-RegistryKey -Key
'HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce'
```

EXAMPLE 2

```
Remove-RegistryKey -Key 'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Run' -
Name 'RunAppInstall'
```

EXAMPLE 3

```
Remove-RegistryKey -Key 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Example' -Name
'(Default)'
```

Resolve-Error

SYNOPSIS

Enumerate error record details.

SYNTAX

```
Resolve-Error [[-ErrorRecord] <Array>] [[-Property] <String[]>] [[-
GetErrorRecord]] [[-GetErrorInvocation]] [[-GetErrorException]] [[-
GetErrorInnerException]] [<CommonParameters>]
```

DESCRIPTION

Enumerate an error record, or a collection of error record, properties. By default, the details for the last error will be enumerated.

PARAMETERS

```
-ErrorRecord <Array>
```

The error record to resolve. The default error record is the latest one: $global:Error[0]. This parameter will also accept an array of error records.

```
-Property <String[]>
```

The list of properties to display from the error record. Use "*" to display all properties.

Default list of error properties is:

- Message
- FullyQualifiedErrorId
- ScriptStackTrace
- PositionMessageInnerException

```
-GetErrorRecord [<SwitchParameter>]
```

Get error record details as represented by $_.

```
-GetErrorInvocation [<SwitchParameter>]
```

Get error record invocation information as represented by $_.InvocationInfo.

```
-GetErrorException [<SwitchParameter>]
```

Get error record exception details as represented by $_.Exception.

```
-GetErrorInnerException [<SwitchParameter>]
```

Get error record inner exception details as represented by $_.Exception.InnerException. Will retrieve all inner exceptions if there is more than one.

EXAMPLE 1

```
Resolve-Error
```

EXAMPLE 2

```
Resolve-Error -Property *
```

EXAMPLE 3

```
Resolve-Error -Property InnerException
```

EXAMPLE 4

```
Resolve-Error -GetErrorInvocation:$false
```

Send-Keys

SYNOPSIS

Send a sequence of keys to one or more application windows.

SYNTAX

```
Send-Keys [[-WindowTitle] <String>] [[-GetAllWindowTitles]] [[-WindowHandle]
<IntPtr>] [[-Keys] <String>] [[-WaitSeconds] <Int32>] [<CommonParameters>]
```

DESCRIPTION

Send a sequence of keys to one or more application window. If window title searched for returns more than one window, then all of them will receive the sent keys.
Function does not work in SYSTEM context unless launched with "psexec.exe -s -i" to run it as an interactive process under the SYSTEM account.

PARAMETERS

```
-WindowTitle <String>
```

The title of the application window to search for using regex matching.

```
-GetAllWindowTitles [<SwitchParameter>]
```

Get titles for all open windows on the system.

```
-WindowHandle <IntPtr>
```

Send keys to a specific window where the Window Handle is already known.

```
-Keys <String>
```

The sequence of keys to send. Info on Key input at: http://msdn.microsoft.com/en-us/library/System.Windows.Forms.SendKeys(v=vs.100).aspx

```
-WaitSeconds <Int32>
```

An optional number of seconds to wait after the sending of the keys.

EXAMPLE 1

```
Send-Keys -WindowTitle 'foobar - Notepad' -Key 'Hello world'
```

Send the sequence of keys "Hello world" to the application titled "foobar - Notepad".

EXAMPLE 2

```
Send-Keys -WindowTitle 'foobar - Notepad' -Key 'Hello world' -WaitSeconds 5
```

Send the sequence of keys "Hello world" to the application titled "foobar - Notepad" and wait 5 seconds.

EXAMPLE 3

```
Send-Keys -WindowHandle ([IntPtr]17368294) -Key 'Hello world'
```

Send the sequence of keys "Hello world" to the application with a Window Handle of '17368294'.

Set-ActiveSetup

SYNOPSIS

Creates an Active Setup entry in the registry to execute a file for each user upon login.

SYNTAX

```
Set-ActiveSetup -StubExePath <String> [-Arguments <String>] [-**DESCRIPTION**
<String>] [-Key <String>] [-Version <String>] [-Locale <String>] [-
DisableActiveSetup] [-ContinueOnError <Boolean>]
```

```
Set-ActiveSetup [-Key <String>] -PurgeActiveSetupKey [-ContinueOnError
<Boolean>] [<CommonParameters>]
```

DESCRIPTION

Active Setup allows handling of per-user changes registry/file changes upon login.
A registry key is created in the HKLM registry hive which gets replicated to the HKCU hive when a user logs in.

If the "Version" value of the Active Setup entry in HKLM is higher than the version value in HKCU, the file referenced in "StubPath" is executed.

This Function:

- Creates the registry entries in HKLM:SOFTWARE\Microsoft\Active Setup\Installed Components$installName.
- Creates StubPath value depending on the file extension of the $StubExePath parameter.
- Handles Version value with YYYYMMDDHHMMSS granularity to permit re-installs on the same day and still trigger Active Setup after Version increase.
- Copies/overwrites the StubPath file to $StubExePath destination path if file exists in 'Files' subdirectory of script directory.
- Executes the StubPath file for the current user as long as not in Session 0 (no need to logout/login to trigger Active Setup).

PARAMETERS

```
-StubExePath <String>
```

Full destination path to the file that will be executed for each user that logs in.
If this file exists in the 'Files' subdirectory of the script directory, it will be copied to the destination path.

```
-Arguments <String>
```

Arguments to pass to the file being executed.

```
-Description <String>
```

Description for the Active Setup. Users will see "Setting up personalized settings for: $Description" at logon. Default is: $installName.

```
-Key <String>
```

Name of the registry key for the Active Setup entry. Default is: $installName.

```
-Version <String>
```

Optional. Specify version for Active setup entry. Active Setup is not triggered if Version value has more than 8 consecutive digits. Use commas to get around this limitation.

```
-Locale <String>
```

Optional. Arbitrary string used to specify the installation language of the file being executed. Not replicated to HKCU.

```
-DisableActiveSetup [<SwitchParameter>]
```

Disables the Active Setup entry so that the StubPath file will not be executed.

```
-PurgeActiveSetupKey [<SwitchParameter>]
```

Remove Active Setup entry from HKLM registry hive. Will also load each logon user's HKCU registry hive to remove Active Setup entry.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Set-ActiveSetup -StubExePath 'C:\Users\Public\Company\ProgramUserConfig.vbs' -
Arguments '/Silent' -Description 'Program User Config' -Key 'ProgramUserConfig'
-Locale 'en'
```

EXAMPLE 2

```
Set-ActiveSetup -StubExePath "$envWinDir\regedit.exe" -Arguments "/S
`"%SystemDrive%\Program Files (x86)\PS App Deploy\PSAppDeployHKCUSettings.reg`""
-Description 'PS App Deploy Config'
```

-Key 'PS_App_Deploy_Config' -ContinueOnError $true

EXAMPLE 3

```
Set-ActiveSetup -Key 'ProgramUserConfig' -PurgeActiveSetupKey
```

Deletes "ProgramUserConfig" active setup entry from all registry hives.

Set-IniValue

SYNOPSIS

Opens an INI file and sets the value of the specified section and key.

SYNTAX

```
Set-IniValue [-FilePath] <String> [-Section] <String> [-Key] <String> [-Value]
<Object> [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Opens an INI file and sets the value of the specified section and key.

PARAMETERS

```
-FilePath <String>
```

Path to the INI file.

```
-Section <String>
```

Section within the INI file.

```
-Key <String>
```

Key within the section of the INI file.

```
-Value <Object>
```

Value for the key within the section of the INI file. To remove a value, set this variable to $null.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Set-IniValue -FilePath "$envProgramFilesX86\IBM\Notes\notes.ini" -Section
'Notes' -Key 'KeyFileName' -Value 'MyFile.ID'
```

Set-PinnedApplication

SYNOPSIS

Pins or unpins a shortcut to the start menu or task bar.

SYNTAX

```
Set-PinnedApplication [-Action] <String> [-FilePath] <String>
[<CommonParameters>]
```

DESCRIPTION

Pins or unpins a shortcut to the start menu or task bar.
This should typically be run in the user context, as pinned items are stored in the user profile.

PARAMETERS

```
-Action <String>
```

Action to be performed. Options:
'PintoStartMenu','UnpinfromStartMenu','PintoTaskbar','UnpinfromTaskbar'.

```
-FilePath <String>
```

Path to the shortcut file to be pinned or unpinned.

EXAMPLE 1

```
Set-PinnedApplication -Action 'PintoStartMenu' -FilePath
"$envProgramFilesX86\IBM\Lotus\Notes\notes.exe"
```

EXAMPLE 2

```
Set-PinnedApplication -Action 'UnpinfromTaskbar' -FilePath
"$envProgramFilesX86\IBM\Lotus\Notes\notes.exe"
```

Set-RegistryKey

SYNOPSIS

Creates a registry key name, value, and value data; it sets the same if it already exists.

SYNTAX

```
Set-RegistryKey [-Key] <String> [[-Name] <String>] [[-Value] <Object>] [[-Type]
{Unknown | String | ExpandString | Binary | DWord | MultiString | QWord | None}]
[[-SID] <String>] [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Creates a registry key name, value, and value data; it sets the same if it already exists.

PARAMETERS

```
-Key <String>
```

The registry key path.

```
-Name <String>
```

The value name.

```
-Value <Object>
```

The value data.

```
-Type
```

The type of registry value to create or set. Options:

- Binary
- DWord
- ExpandString
- MultiString
- None
- QWord
- String (Default)
- Unknown

Dword should be specified as a decimal.

```
-SID <String>
```

The security identifier (SID) for a user. Specifying this parameter will convert a HKEY_CURRENT_USER registry key to the HKEY_USERS$SID format.

Specify this parameter from the Invoke-HKCURegistrySettingsForAllUsers function to read/edit HKCU registry settings for all users on the system.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Set-RegistryKey -Key $blockedAppPath -Name 'Debugger' -Value
$blockedAppDebuggerValue
```

EXAMPLE 2

```
Set-RegistryKey -Key 'HKEY_LOCAL_MACHINE\SOFTWARE' -Name 'Application' -Type
'Dword' -Value '1'
```

EXAMPLE 3

```
Set-RegistryKey -Key
'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce' -Name
'Debugger' -Value $blockedAppDebuggerValue -Type String
```

EXAMPLE 4

```
Set-RegistryKey -Key 'HKCU\Software\Microsoft\Example' -Name 'Data' -Value
(0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x02,0x01,0x01,0x01
,0x01,0x01,0x01,0x01,0x02,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x00,0x01,0x01,0x01
,0x02,0x02,0x02) -Type 'Binary'
```

EXAMPLE 5

```
Set-RegistryKey -Key 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Example' -Name
'(Default)' -Value "Text"
```

Set-ServiceStartMode

SYNOPSIS

Set the service startup mode.

SYNTAX

```
Set-ServiceStartMode [-Name] <String> [[-ComputerName] <String>] [-StartMode]
<String> [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Set the service startup mode.

PARAMETERS

```
-Name <String>
```

Specify the name of the service.

```
-ComputerName <String>
```

Specify the name of the computer. Default is: the local computer.

```
-StartMode <String>
```

Specify startup mode for the service. Options: Automatic, Automatic (Delayed Start), Manual, Disabled, Boot, System.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Set-ServiceStartMode -Name 'wuauserv' -StartMode 'Automatic (Delayed Start)'
```

Set-Shortcut

SYNOPSIS

Modifies a .lnk or .url type shortcut

SYNTAX

```
Set-Shortcut [-Path] <String> [-TargetPath] <String> [[-Arguments] <String>] [[-
IconLocation] <String>] [[-IconIndex] <String>] [[-Description] <String>] [[-
WorkingDirectory] <String>] [[-WindowStyle] <String>] [-RunAsAdmin] [[-Hotkey]
<String>] [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Modifies a shortcut - .lnk or .url file, with configurable options.
Only specify the parameters that you want to change.

PARAMETERS

```
-Path <String>
```

Path to the shortcut to be changed.

```
-TargetPath <String>
```

Changes target path or URL that the shortcut launches.

```
-Arguments <String>
```

Changes Arguments to be passed to the target path.

```
-IconLocation <String>
```

Changes location of the icon used for the shortcut.

```
-IconIndex <String>
```

Change the index of the icon. Executables, DLLs, ICO files with multiple icons need the icon index to be specified. This parameter is an Integer. The first index is 0.

```
-Description <String>
```

Changes description of the shortcut.

```
-WorkingDirectory <String>
```

Changes Working Directory to be used for the target path.

```
-WindowStyle <String>
```

Changes the Windows style of the application. Options: Normal, Maximized, Minimized, DontChange. Default is: DontChange.

```
-RunAsAdmin [<SwitchParameter>]
```

Set shortcut to run program as administrator. This option will prompt user to elevate when executing shortcut. If not specified or set to $null, the flag will not be changed.

```
-Hotkey <String>
```

Changes the Hotkey to launch the shortcut, e.g. "CTRL+SHIFT+F".

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
New-Shortcut -Path "$envProgramData\Microsoft\Windows\Start Menu\My
Shortcut.lnk" -TargetPath "$envWinDir\system32\notepad.exe" -IconLocation
"$envWinDir\system32\notepad.exe" -Description 'Notepad' -WorkingDirectory
"$envHomeDrive\$envHomePath"
```

NOTES

Url shortcuts only support TargetPath, IconLocation and IconIndex. Other parameters are ignored.

Show-BalloonTip

SYNOPSIS

Displays a balloon tip notification in the system tray.

SYNTAX

```
Show-BalloonTip [-BalloonTipText] <String> [[-BalloonTipTitle] <String>] [[-
BalloonTipIcon] {None | Info | Warning | Error}] [[-BalloonTipTime] <Int32>]
[<CommonParameters>]
```

DESCRIPTION

Displays a balloon tip notification in the system tray.

PARAMETERS

```
-BalloonTipText <String>
```

Text of the balloon tip.

```
-BalloonTipTitle <String>
```

Title of the balloon tip.

```
-BalloonTipIcon
```

Icon to be used. Options: 'Error', 'Info', 'None', 'Warning'. Default is: Info.

```
-BalloonTipTime <Int32>
```

Time in milliseconds to display the balloon tip. Default: 10000.

```
-NoWait <Boolean>
```

Create the balloontip asynchronously. Default: $false

EXAMPLE 1

```
Show-BalloonTip -BalloonTipText 'Installation Started' -BalloonTipTitle
'Application Name'
```

EXAMPLE 2

```
Show-BalloonTip -BalloonTipIcon 'Info' -BalloonTipText 'Installation Started' -
BalloonTipTitle 'Application Name' -BalloonTipTime 1000
```

Show-DialogBox

SYNOPSIS

Display a custom dialog box with optional title, buttons, icon and timeout.

Show-InstallationPrompt is recommended over this function as it provides more customization and uses consistent branding with the other UI components.

SYNTAX

```
Show-DialogBox [-Text] <String> [-Title <String>] [-Buttons <String>] [-
DefaultButton <String>] [-Icon <String>] [-Timeout <String>] [-TopMost
<Boolean>] [<CommonParameters>]
```

DESCRIPTION

Display a custom dialog box with optional title, buttons, icon and timeout. The default button is "OK", the default Icon is "None", and the default Timeout is none.

PARAMETERS

```
-Text <String>
```

Text in the message dialog box

```
-Title <String>
```

Title of the message dialog box

```
-Buttons <String>
```

Buttons to be included on the dialog box. Options:

- OK (Default)
- OKCancel
- AbortRetryIgnore
- YesNoCancel
- YesNo
- RetryCancel
- CancelTryAgainContinue

```
-DefaultButton <String>
```

The Default button that is selected. Options:

- First (Default)
- Second
- Third

```
-Icon <String>
```

Icon to display on the dialog box. Options: None, Stop, Question, Exclamation, Information. Default: None.

```
-Timeout <String>
```

Timeout period in seconds before automatically closing the dialog box with the return message "Timeout". Default: UI timeout value set in the config XML file.

```
-TopMost <Boolean>
```

Specifies whether the message box is a system modal message box and appears in a topmost window. Default: $true.

EXAMPLE 1

```
Show-DialogBox -Title 'Installed Complete' -Text 'Installation has completed.
Please click OK and restart your computer.' -Icon 'Information'
```

EXAMPLE 2

```
Show-DialogBox -Title 'Installation Notice' -Text 'Installation will take
approximately 30 minutes. Do you wish to proceed?' -Buttons 'OKCancel' -
DefaultButton 'Second' -Icon 'Exclamation' -Timeout 600 -Topmost $false
```

Show-InstallationProgress

SYNOPSIS

Displays a progress dialog in a separate thread with an updateable custom message.

SYNTAX

```
Show-InstallationProgress [[-StatusMessage] <String>] [[-WindowLocation]
<String>] [[-TopMost] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Create a WPF window in a separate thread to display a marquee style progress ellipse with a custom message that can be updated. The status message supports line breaks. The first time this function is called in a script, it will display a balloon tip notification to indicate that the installation has started (provided balloon tips are enabled in the configuration).

PARAMETERS

```
-StatusMessage <String>
```

The status message to be displayed. The default status message is taken from the XML configuration file.

```
-WindowLocation <String>
```

The location of the progress window. Default: just below top, centered.

```
-TopMost <Boolean>
```

Specifies whether the progress window should be topmost. Default: $true.

EXAMPLE 1

```
Show-InstallationProgress
```

Uses the default status message from the XML configuration file.

EXAMPLE 2

```
Show-InstallationProgress -StatusMessage 'Installation in Progress...'
```

EXAMPLE 3

```
Show-InstallationProgress -StatusMessage "Installation in Progress...`nThe
installation may take 20 minutes to complete."
```

EXAMPLE 4

```
Show-InstallationProgress -StatusMessage 'Installation in Progress...' -
WindowLocation 'BottomRight' -TopMost $false
```

Show-InstallationPrompt

SYNOPSIS

Displays a custom installation prompt with the toolkit branding and optional buttons.

SYNTAX

```
Show-InstallationPrompt [[-Title] <String>] [[-Message] <String>] [[-
MessageAlignment] <String>] [[-ButtonRightText] <String>] [[-ButtonLeftText]
<String>] [[-ButtonMiddleText] <String>] [[-Icon] <String>] [-NoWait] [-
PersistPrompt] [[-MinimizeWindows] <Boolean>] [[-Timeout] <Int32>] [[-
ExitOnTimeout] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Any combination of Left, Middle or Right buttons can be displayed. The return value of the button clicked by the user is the button text specified.

PARAMETERS

```
-Title <String>
```

Title of the prompt. Default: the application installation name.

```
-Message <String>
```

Message text to be included in the prompt

```
-MessageAlignment <String>
```

Alignment of the message text. Options: Left, Center, Right. Default: Center.

```
-ButtonRightText <String>
```

Show a button on the right of the prompt with the specified text

```
-ButtonLeftText <String>
```

Show a button on the left of the prompt with the specified text

```
-ButtonMiddleText <String>
```

Show a button in the middle of the prompt with the specified text

```
-Icon <String>
```

Show a system icon in the prompt. Options:

- Application
- Asterisk
- Error
- Exclamation
- Hand
- Information
- None
- Question
- Shield
- Warning
- WinLogo

```
-NoWait [<SwitchParameter>]
```

Specifies whether to show the prompt asynchronously (i.e. allow the script to continue without waiting for a response). Default: $false.

```
-PersistPrompt [<SwitchParameter>]
```

Specify whether to make the prompt persist in the center of the screen every 10 seconds. The user will have no option but to respond to the prompt - resistance is futile!

```
-MinimizeWindows <Boolean>
```

Specifies whether to minimize other windows when displaying prompt. Default: $false.

```
-Timeout <Int32>
```

Specifies the time period in seconds after which the prompt should timeout. Default: UI timeout value set in the config XML file.

```
-ExitOnTimeout <Boolean>
```

Specifies whether to exit the script if the UI times out. Default: $true.

EXAMPLE 1

```
Show-InstallationPrompt -Message 'Do you want to proceed with the installation?'
-ButtonRightText 'Yes' -ButtonLeftText 'No'
```

EXAMPLE 2

```
Show-InstallationPrompt -Title 'Funny Prompt' -Message 'How are you feeling
today?' -ButtonRightText 'Good' -ButtonLeftText 'Bad' -ButtonMiddleText
'Indifferent'
```

EXAMPLE 3

```
Show-InstallationPrompt -Message 'You can customize text to appear at the end of
an install, or remove it completely for unattended installations.' -Icon
Information -NoWait
```

Show-InstallationRestartPrompt

SYNOPSIS

Displays a restart prompt with a countdown to a forced restart.

SYNTAX

```
Show-InstallationRestartPrompt [[-CountdownSeconds] <Int32>] [[-
CountdownNoHideSeconds] <Int32>] [-NoCountdown] [<CommonParameters>]
```

DESCRIPTION

Displays a restart prompt with a countdown to a forced restart.

PARAMETERS

```
-CountdownSeconds <Int32>
```

Specifies the number of seconds to countdown before the system restart.

```
-CountdownNoHideSeconds <Int32>
```

Specifies the number of seconds to display the restart prompt without allowing the window to be hidden.

```
-NoCountdown [<SwitchParameter>]
```

Specifies not to show a countdown, just the Restart Now and Restart Later buttons. The UI will restore/reposition itself persistently based on the interval value specified in the config file.

EXAMPLE 1

```
Show-InstallationRestartPrompt -Countdownseconds 600 -CountdownNoHideSeconds 60
```

EXAMPLE 2

```
Show-InstallationRestartPrompt -NoCountdown
```

Show-InstallationWelcome

SYNOPSIS

Show a welcome dialog prompting the user with information about the installation and actions to be performed before the installation can begin.

SYNTAX

```
Show-InstallationWelcome [-CloseApps <String>] [-Silent] [-CloseAppsCountdown
<Int32>] [-ForceCloseAppsCountdown <Int32>] [-PromptToSave] [-PersistPrompt] [-
BlockExecution] [-AllowDefer] [-AllowDeferCloseApps] [-DeferTimes <Int32>] [-
DeferDays <Int32>] [-DeferDeadline <String>] [-MinimizeWindows <Boolean>] [-
TopMost <Boolean>] [-ForceCountdown <Int32>] [-CustomText] [<CommonParameters>]
```

```
Show-InstallationWelcome [-CloseApps <String>] [-Silent] [-CloseAppsCountdown
<Int32>] [-ForceCloseAppsCountdown <Int32>] [-PromptToSave] [-PersistPrompt] [-
BlockExecution] [-AllowDefer] [-AllowDeferCloseApps] [-DeferTimes <Int32>] [-
DeferDays <Int32>] [-DeferDeadline <String>] -CheckDiskSpace [-RequiredDiskSpace
<Int32>] [-MinimizeWindows <Boolean>] [-TopMost <Boolean>] [-ForceCountdown
<Int32>] [-CustomText] [<CommonParameters>]
```

DESCRIPTION

The following prompts can be included in the welcome dialog:

a) Close the specified running applications, or optionally close the applications without showing a prompt (using the -Silent switch).
b) Defer the installation a certain number of times, for a certain number of days or until a deadline is reached.
c) Countdown until applications are automatically closed.
d) Prevent users from launching the specified applications while the installation is in progress.

Notes:

The process descriptions are retrieved from WMI, with a fall back on the process name if no description is available. Alternatively, you can specify the description yourself with a '=' symbol - see examples.

The dialog box will timeout after the timeout specified in the XML configuration file (default 1 hour and 55 minutes) to prevent SCCM installations from timing out and returning a failure code to SCCM. When the dialog times out, the script will exit and return a 1618 code (SCCM fast retry code).

PARAMETERS

```
-CloseApps <String>
```

Name of the process to stop (do not include the .exe). Specify multiple processes separated by a comma. Specify custom descriptions like this: "winword=Microsoft Office Word,excel=Microsoft Office Excel"

```
-Silent [<SwitchParameter>]
```

Stop processes without prompting the user.

```
-CloseAppsCountdown <Int32>
```

Option to provide a countdown in seconds until the specified applications are automatically closed. This only takes effect if deferral is not allowed or has expired.

```
-ForceCloseAppsCountdown <Int32>
```

Option to provide a countdown in seconds until the specified applications are automatically closed regardless of whether deferral is allowed.

```
-PromptToSave [<SwitchParameter>]
```

Specify whether to prompt to save working documents when the user chooses to close applications by selecting the "Close Programs" button. Option does not work in SYSTEM context unless toolkit launched with "psexec.exe -s -i" to run it as an interactive process under the SYSTEM account.

```
-PersistPrompt [<SwitchParameter>]
```

Specify whether to make the prompt persist in the center of the screen every 10 seconds. The user will have no option but to respond to the prompt. This only takes effect if deferral is not allowed or has expired.

```
-BlockExecution [<SwitchParameter>]
```

Option to prevent the user from launching the process/application during the installation.

```
-AllowDefer [<SwitchParameter>]
```

Enables an optional defer button to allow the user to defer the installation.

```
-AllowDeferCloseApps [<SwitchParameter>]
```

Enables an optional defer button to allow the user to defer the installation only if there are running applications that need to be closed.

```
-DeferTimes <Int32>
```

Specify the number of times the installation can be deferred.

```
-DeferDays <Int32>
```

Specify the number of days since first run that the installation can be deferred. This is converted to a deadline.

```
-DeferDeadline <String>
```

Specify the deadline date until which the installation can be deferred.

Specify the date in the local culture if the script is intended for that same culture.

If the script is intended to run on EN-US machines, specify the date in the format: "08/25/2013" or "08-25-2013" or "08-25-2013 18:00:00"

If the script is intended for multiple cultures, specify the date in the universal sortable date/time format: "2013-08-22 11:51:52Z"

The deadline date will be displayed to the user in the format of their culture.

```
-CheckDiskSpace [<SwitchParameter>]
```

Specify whether to check if there is enough disk space for the installation to proceed.

If this parameter is specified without the RequiredDiskSpace parameter, the required disk space is calculated automatically based on the size of the script source and associated files.

```
-RequiredDiskSpace <Int32>
```

Specify required disk space in MB, used in combination with CheckDiskSpace.

```
-MinimizeWindows <Boolean>
```

Specifies whether to minimize other windows when displaying prompt. Default: $true.

```
-TopMost <Boolean>
```

Specifies whether the windows is the topmost window. Default: $true.

```
-ForceCountdown <Int32>
```

Specify a countdown to display before automatically proceeding with the installation when a deferral is enabled.

```
-CustomText [<SwitchParameter>]
```

Specify whether to display a custom message specified in the XML file. Custom message must be populated for each language section in the XML.

EXAMPLE 1

```
Show-InstallationWelcome -CloseApps 'iexplore,winword,excel'
```

Prompt the user to close Internet Explorer, Word and Excel.

EXAMPLE 2

```
Show-InstallationWelcome -CloseApps 'winword,excel' -Silent
```

Close Word and Excel without prompting the user.

EXAMPLE 3

```
Show-InstallationWelcome -CloseApps 'winword,excel' -BlockExecution
```

Close Word and Excel and prevent the user from launching the applications while the installation is in progress.

EXAMPLE 4

```
Show-InstallationWelcome -CloseApps 'winword=Microsoft Office
Word,excel=Microsoft Office Excel' -CloseAppsCountdown 600
```

Prompt the user to close Word and Excel, with customized descriptions for the applications and automatically close the applications after 10 minutes.

EXAMPLE 5

```
Show-InstallationWelcome -CloseApps 'winword,msaccess,excel' -PersistPrompt
```

Prompt the user to close Word, MSAccess and Excel.

By using the PersistPrompt switch, the dialog will return to the center of the screen every 10 seconds so the user cannot ignore it by dragging it aside.

EXAMPLE 6

```
Show-InstallationWelcome -AllowDefer -DeferDeadline '25/08/2013'
```

Allow the user to defer the installation until the deadline is reached.

—————————————— EXAMPLE 7 ——————————————

```
Show-InstallationWelcome -CloseApps 'winword,excel' -BlockExecution -AllowDefer
-DeferTimes 10 -DeferDeadline '25/08/2013' -CloseAppsCountdown 600
```

Close Word and Excel and prevent the user from launching the applications while the installation is in progress.
Allow the user to defer the installation a maximum of 10 times or until the deadline is reached, whichever happens first.
When deferral expires, prompt the user to close the applications and automatically close them after 10 minutes.

Start-ServiceAndDependencies

SYNOPSIS

Start Windows service and its dependencies.

SYNTAX

```
Start-ServiceAndDependencies [-Name] <String> [[-ComputerName] <String>] [-
SkipServiceExistsTest] [-SkipDependentServices] [[-PendingStatusWait]
<TimeSpan>] [-PassThru] [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Start Windows service and its dependencies.

PARAMETERS

```
-Name <String>
```

Specify the name of the service.

```
-ComputerName <String>
```

Specify the name of the computer. Default is: the local computer.

```
-SkipServiceExistsTest [<SwitchParameter>]
```

Choose to skip the test to check whether or not the service exists if it was already done outside of this function.

```
-SkipDependentServices [<SwitchParameter>]
```

Choose to skip checking for and starting dependent services. Default is: $false.

```
-PendingStatusWait <TimeSpan>
```

The amount of time to wait for a service to get out of a pending state before continuing. Default is 60 seconds.

```
-PassThru [<SwitchParameter>]
```

Return the System.ServiceProcess.ServiceController service object.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Start-ServiceAndDependencies -Name 'wuauserv'
```

Stop-ServiceAndDependencies

SYNOPSIS

Stop Windows service and its dependencies.

SYNTAX

```
Stop-ServiceAndDependencies [-Name] <String> [[-ComputerName] <String>] [-
SkipServiceExistsTest] [-SkipDependentServices] [[-PendingStatusWait]
<TimeSpan>] [-PassThru] [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Stop Windows service and its dependencies.

PARAMETERS

```
-Name <String>
```

Specify the name of the service.

```
-ComputerName <String>
```

Specify the name of the computer. Default is: the local computer.

```
-SkipServiceExistsTest [<SwitchParameter>]
```

Choose to skip the test to check whether or not the service exists if it was already done outside of this function.

```
-SkipDependentServices [<SwitchParameter>]
```

Choose to skip checking for and stopping dependent services. Default is: $false.

```
-PendingStatusWait <TimeSpan>
```

The amount of time to wait for a service to get out of a pending state before continuing. Default is 60 seconds.

```
-PassThru [<SwitchParameter>]
```

Return the System.ServiceProcess.ServiceController service object.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Stop-ServiceAndDependencies -Name 'wuauserv'
```

Test-Battery

SYNOPSIS

Tests whether the local machine is running on AC power or not.

SYNTAX

```
Test-Battery [-PassThru] [<CommonParameters>]
```

DESCRIPTION

Tests whether the local machine is running on AC power and returns true/false. For detailed information, use -PassThru option.

PARAMETERS

```
-PassThru [<SwitchParameter>]
```

Outputs a hashtable containing the following properties:

- IsLaptop

- IsUsingACPower
- ACPowerLineStatus
- BatteryChargeStatus
- BatteryLifePercent
- BatteryLifeRemaining
- BatteryFullLifetime

EXAMPLE 1

```
Test-Battery
```

EXAMPLE 2

```
(Test-Battery -PassThru).IsLaptop
```

Determines if the current system is a laptop or not.

Test-MSUpdates

SYNOPSIS

Test whether a Microsoft Windows update is installed.

SYNTAX

```
Test-MSUpdates [-KBNumber] <String> [[-ContinueOnError] <Boolean>]
[<CommonParameters>]
```

DESCRIPTION

Test whether a Microsoft Windows update is installed.

PARAMETERS

```
-KBNumber <String>
```

KB Number of the update.

```
-ContinueOnError <Boolean>
```

Suppress writing log message to console on failure to write message to log file. Default is: $true.

EXAMPLE 1

```
Test-MSUpdates -KBNumber 'KB2549864'
```

Test-NetworkConnection

SYNOPSIS

Tests for an active local network connection, excluding wireless and virtual network adapters.

SYNTAX

```
Test-NetworkConnection [<CommonParameters>]
```

DESCRIPTION

Tests for an active local network connection, excluding wireless and virtual network adapters, by querying the Win32_NetworkAdapter WMI class.

EXAMPLE 1

```
Test-NetworkConnection
```

Test-PowerPoint

SYNOPSIS

Tests whether PowerPoint is running in either fullscreen slideshow mode or presentation mode.

SYNTAX

```
Test-PowerPoint [<CommonParameters>]
```

DESCRIPTION

Tests whether someone is presenting using PowerPoint in either fullscreen slideshow mode or presentation mode.

EXAMPLE 1

```
Test-PowerPoint
```

Test-ServiceExists

SYNOPSIS

Check to see if a service exists.

SYNTAX

```
Test-ServiceExists [-Name] <String> [[-ComputerName] <String>] [-PassThru] [[-
ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Check to see if a service exists (using WMI method because Get-Service will generate ErrorRecord if service doesn't exist).

PARAMETERS

```
-Name <String>
```

Specify the name of the service.

Note: Service name can be found by executing "Get-Service | Format-Table -AutoSize -Wrap" or by using the properties screen of a service in services.msc.

```
-ComputerName <String>
```

Specify the name of the computer. Default is: the local computer.

```
-PassThru [<SwitchParameter>]
```

Return the WMI service object.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Test-ServiceExists -Name 'wuauserv'
```

EXAMPLE 2

```
Test-ServiceExists -Name 'testservice' -PassThru | Where-Object { $_ } |
ForEach-Object { $_.Delete() }
```

Check if a service exists and then delete it by using the -PassThru parameter.

Update-Desktop

SYNOPSIS

Refresh the Windows Explorer Shell, which causes the desktop icons and the environment variables to be reloaded.

SYNTAX

```
Update-Desktop [[-ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Refresh the Windows Explorer Shell, which causes the desktop icons and the environment variables to be reloaded.

PARAMETERS

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Update-Desktop
```

NOTES

This function has an alias: Refresh-Desktop

Update-GroupPolicy

SYNOPSIS

Performs a gpupdate command to refresh Group Policies on the local machine.

SYNTAX

```
powershell Update-GroupPolicy [-ContinueOnError] [<Boolean>]
 [<CommonParameters>]
```

DESCRIPTION

Performs a gpupdate command to refresh Group Policies on the local machine.

PARAMETERS

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Update-GroupPolicy
```

Update-SessionEnvironmentVariables

SYNOPSIS

Updates the environment variables for the current PowerShell session with any environment variable changes that may have occurred during script execution.

SYNTAX

```
Update-SessionEnvironmentVariables [-LoadLoggedOnUserEnvironmentVariables] [[-
ContinueOnError] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Environment variable changes that take place during script execution are not visible to the current PowerShell session.

Use this function to refresh the current PowerShell session with all environment variable settings.

PARAMETERS

```
-LoadLoggedOnUserEnvironmentVariables [<SwitchParameter>]
```

If script is running in SYSTEM context, this option allows loading environment variables from the active console user. If no console user exists but users are logged in, such as on

terminal servers, then the first logged-in non-console user.

```
-ContinueOnError <Boolean>
```

Continue if an error is encountered. Default is: $true.

EXAMPLE 1

```
Update-SessionEnvironmentVariables
```

NOTES

This function has an alias: Refresh-SessionEnvironmentVariables

---

Write-Log

SYNOPSIS

Write messages to a log file in CMTrace.exe compatible format or Legacy text file format.

SYNTAX

```
Write-Log [-Message] <String[]> [[-Severity] <Int16>] [[-Source] <String>] [[-
ScriptSection] <String>] [[-LogType] <String>] [[-LogFileDirectory] <String>]
[[-LogFileName] <String>] [[-MaxLogFileSizeMB] <Decimal>] [[-WriteHost]
<Boolean>] [[-ContinueOnError] <Boolean>] [[-PassThru]] [[-DebugMessage]] [[-
LogDebugMessage] <Boolean>] [<CommonParameters>]
```

DESCRIPTION

Write messages to a log file in CMTrace.exe compatible format or Legacy text file format and optionally display in the console.

PARAMETERS

```
-Message <String[]>
```

The message to write to the log file or output to the console.

```
-Severity <Int16>
```

Defines message type. When writing to console or CMTrace.exe log format, it allows highlighting of message type.
Options:

- 1 - Information (default),
- 2 - Warning (highlighted in yellow)
- 3 - Error (highlighted in red)

```
-Source <String>
```

The source of the message being logged.

```
-ScriptSection <String>
```

The heading for the portion of the script that is being executed. Default is: $script:installPhase.

```
-LogType <String>
```

Choose whether to write a CMTrace.exe compatible log file or a Legacy text log file.

```
-LogFileDirectory <String>
```

Set the directory where the log file will be saved.

```
-LogFileName <String>
```

Set the name of the log file.

```
-MaxLogFileSizeMB <Decimal>
```

Maximum file size limit for log file in megabytes (MB). Default is 10 MB.

```
-WriteHost <Boolean>
```

Write the log message to the console.

```
-ContinueOnError <Boolean>
```

Suppress writing log message to console on failure to write message to log file. Default is: $true.

```
-PassThru [<SwitchParameter>]
```

Return the message that was passed to the function

```
-DebugMessage [<SwitchParameter>]
```

Specifies that the message is a debug message. Debug messages only get logged if -LogDebugMessage is set to $true.

```
-LogDebugMessage <Boolean>
```

Debug messages only get logged if this parameter is set to $true in the config XML file.

EXAMPLE 1

```
Write-Log -Message "Installing patch MS15-031" -Source 'Add-Patch' -LogType
'CMTrace'
```

EXAMPLE 2

```
Write-Log -Message "Script is running on Windows 8" -Source 'Test-ValidOS' -
LogType 'Legacy'
```

EXAMPLE 3

```
Write-Log -Message "Log only message" -WriteHost $false
```