PowerShell App Deployment Toolkit

Administrator Guide

http://psappdeploytoolkit.com

Document Version **3.6.9**

1 Contents

2 Overview

Introduction

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 uninstalling multiple applications, prompting users to close apps, 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 SCCM 2007/2012; provides installation and uninstallation deployment types with options on how to handle exit codes, such as supressing reboots or returning a fast retry code.

Updatable - 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 PowerShell App Deployment Toolkit has been developed (and tested) to work with a wide range of Operating Systems from Windows XP to Windows 8.1 (and the Windows Server equivalents) 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 lifecycle 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 on hundreds of thousands of clients around the globe.

Licensing

The PowerShell App Deployment Toolkit is provided under the Microsoft Public License:

https://psappdeploytoolkit.codeplex.com/license

We have invested a lot of personal time in the creation and ongoing development, maintenance and support of this community tool – 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:

https://psappdeploytoolkit.codeplex.com/wikipage?title=Donate

3   Toolkit Functionality

User Interface

- 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 SCCM task sequence or session 0).

- The UI is localized into several languages and more can be easily added using the XML configuration file.

Functions/Logic

- 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 SCCM 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 slideshow is running in full screen presentation mode

Integration with SCCM

- Handles SCCM exit codes, including time sensitive dialogs supporting SCCM's Fast Retry feature - providing more accurate SCCM Reporting (no more Failed due to timeout errors).

- Ability to prevent reboot codes (3010) from being passed back to SCCM, which would cause a reboot prompt.

- Supports the CM12 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 CM12 application dependency chain.

- Compared to compiled deployment packages, e.g. WiseScript, the Toolkit utilises the SCCM cache correctly and SCCM Distribution Point bandwidth more efficiently by using loose files.

Help Console

- A graphical console for browsing the help documentation for the toolkit functions.

4 Toolkit Components

Toolkit File Structure

Files

The toolkit is comprised of the following files:

Deploy-Application.ps1

Performs the actual install / uninstall and is the only file that needs to be modified, depending on your level of customisation.

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.
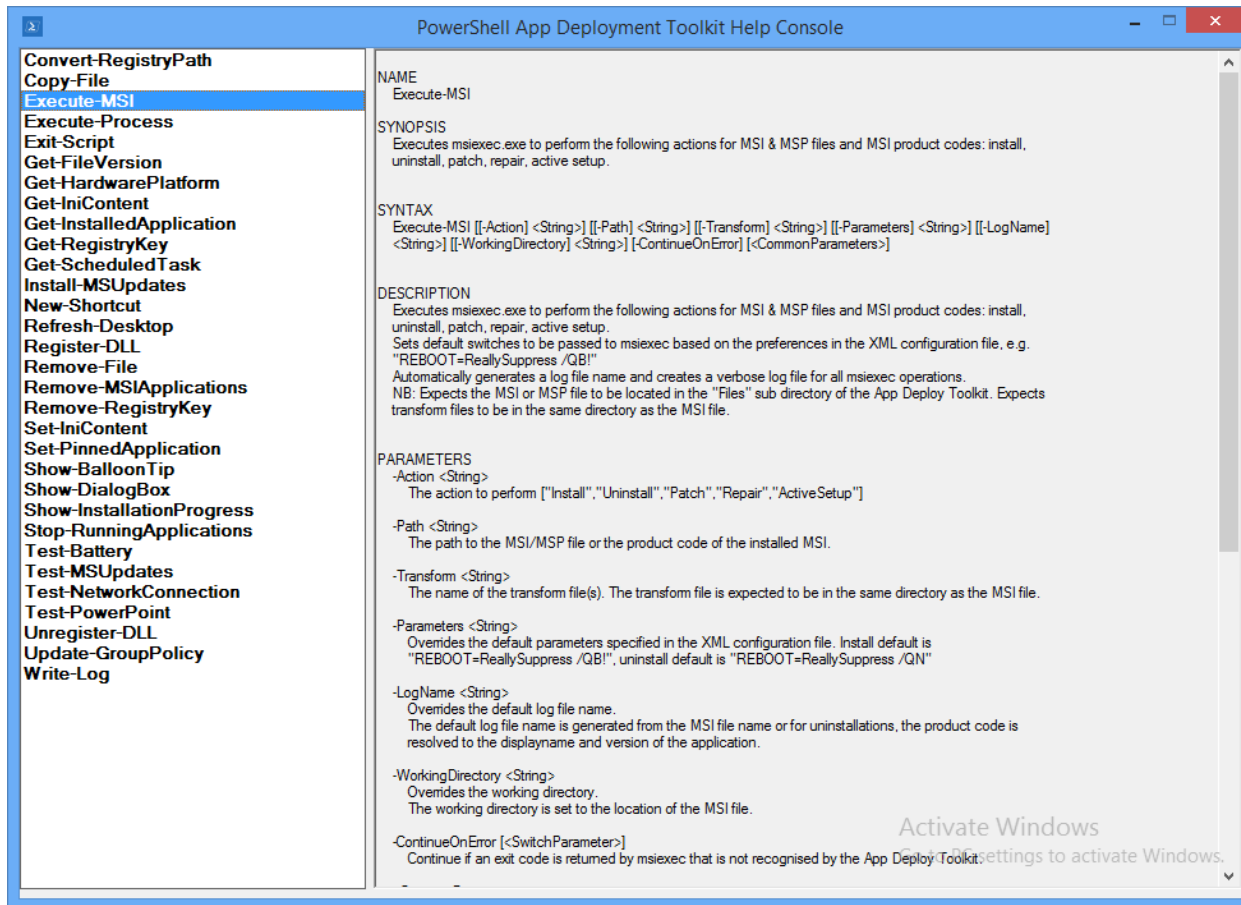
Directories

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.

AppDeployToolkit

Folder containing the toolkit dependency files.

Files

Folder containing your main setup files, e.g. MSI

SupportFiles

Folder containing any supporting files such as 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 localised in 11 different languages: English, French, Spanish, Portuguese, German, Italian, Dutch, Swedish, Danish, Norweigan 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 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.
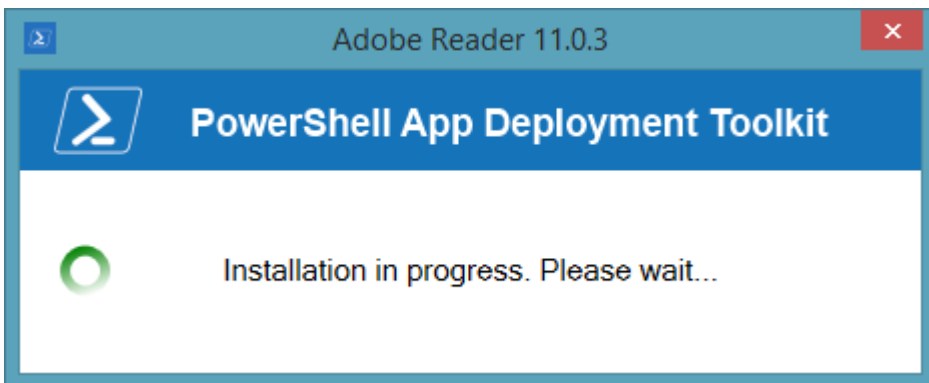
The progress message can be dynamically updated to indicate the stage of the installation or to display custom messages to the user, using the "Show-InstallationProgress" function.
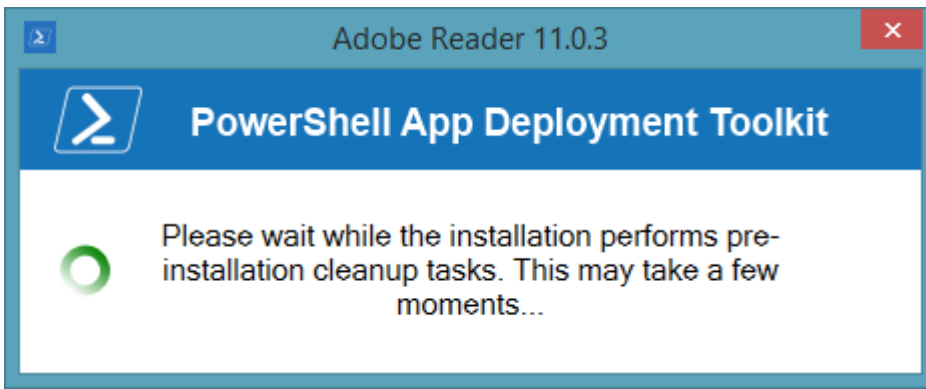
Installation Welcome Prompt

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.
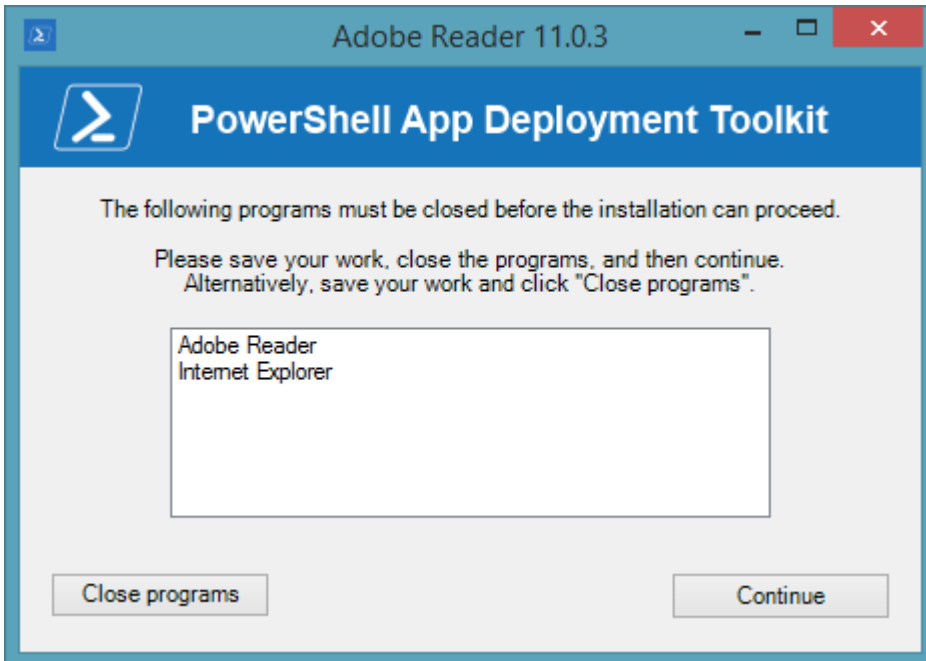
Welcome prompt with close programs option and defer option:
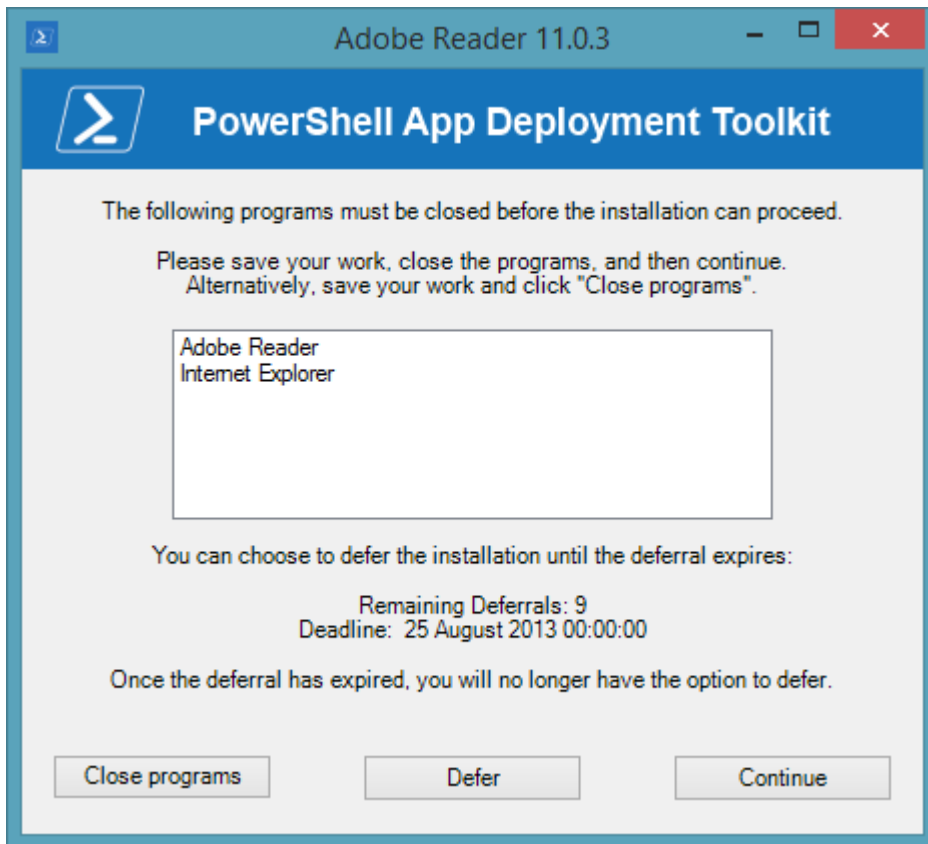
Welcome prompt with close programs options and countdown to automatic closing of applications:
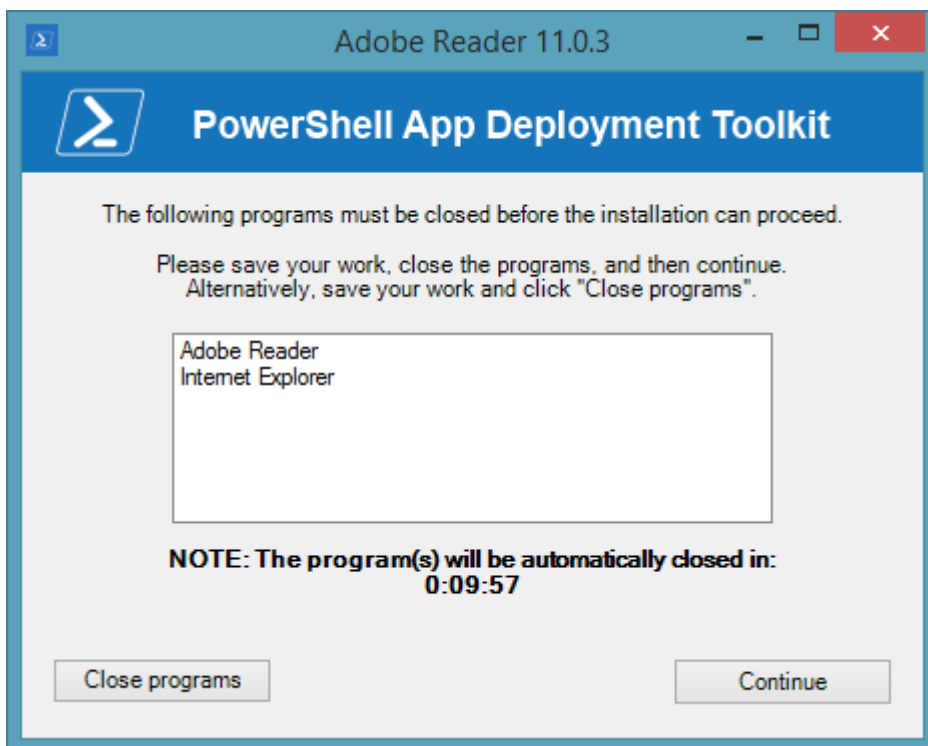
Welcome prompt with just a defer option:

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.
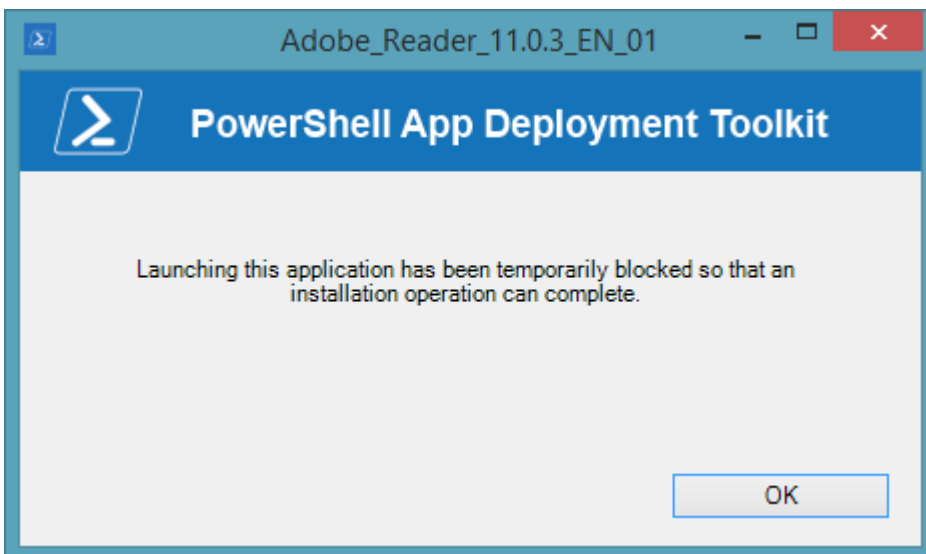
Disk Space Requirements

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.

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.

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.

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.
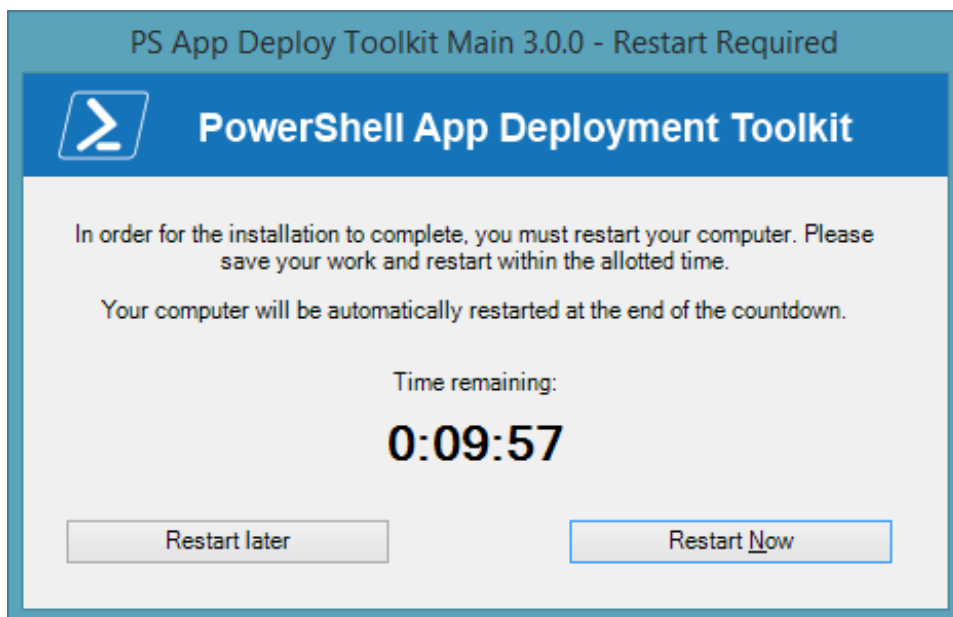
Balloon tip notifications

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.
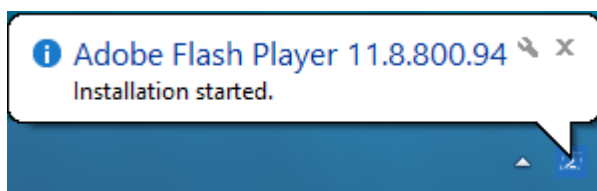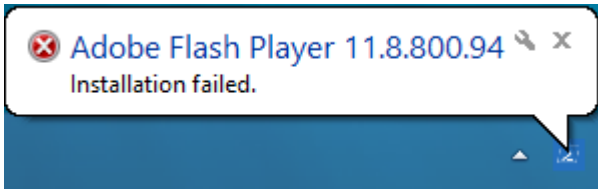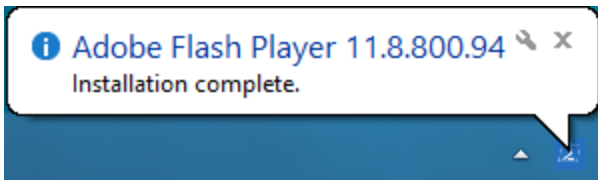
Custom Dialog box

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.
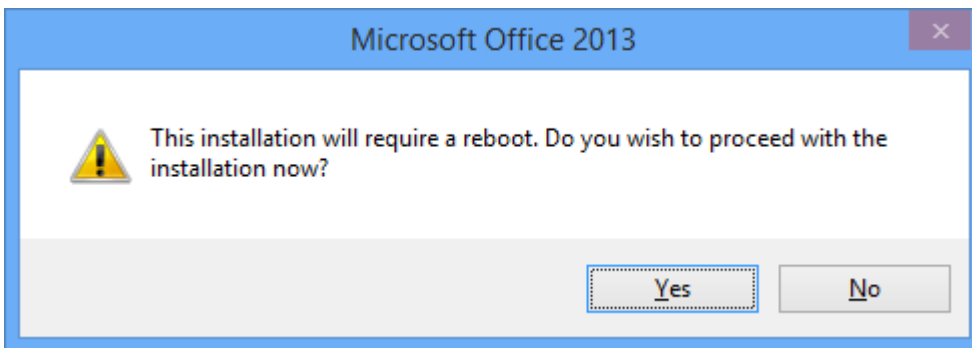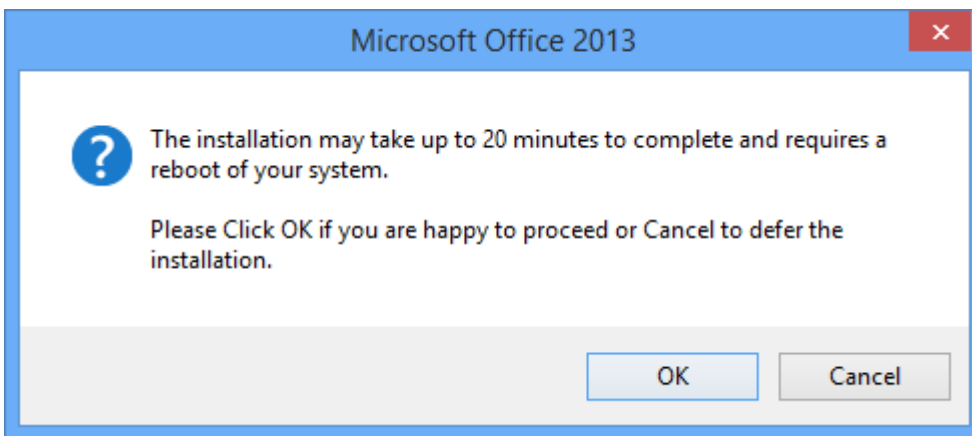
Logging

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**

5   Toolkit Usage

Overview

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

The Deploy-Application.ps1 is broken down into the following sections:

Initialization   e.g. Variables such as App Vendor, App Name, App Version

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

Installation   e.g. Install the primary application, or components of the application

Post-Installation   e.g. Drop additional files, registry tweaks

Uninstallation   e.g. Uninstall/rollback the changes performed in the install section.

Launching the Toolkit

Overview

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

1.   Launch "Deploy-Application.ps1" PowerShell script as administrator.

2.   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-Application.ps1

Deploy an application for installation

Deploy-Application.ps1 -DeploymentType "Uninstall" -DeployMode "Silent"

Deploy an application for uninstallation in silent mode

Deploy-Application.exe /32 -DeploymentType "Uninstall" -DeployMode "Silent"

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

Deploy-Application.exe -AllowRebootPassThru

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

Deploy-Application.exe "Custom-Script.ps1"

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

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

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

Toolkit Parameters

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

-DeploymentType "Install" | "Uninstall" (default is install)

Specify whether to install or uninstall the application.

-DeployMode "Interactive" | "Silent" | "NonInteractive" (default is interactive)

Specify whether the installation should be run in Interactive, Silent or NonInteractive mode.

Interactive = Shows dialogs

Silent = No dialogs (progress and balloon tip notifications are supressed)

NonInteractive = Very silent, i.e. no blocking apps. NonInteractive mode is automatically set if it is detected that the process is not user interactive.

-AllowRebootPassThru $true | $false (default is false)

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

-TerminalServerMode $true | $false (default is false)

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

-DisableLogging (switch parameter, default is false)

Disables logging to file for the script.

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.

CompressLogs (option in AppDeployToolkitConfig.xml) - 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.

Example Deployments

Building an Adobe Reader installation with the PowerShell App Deployment Toolkit

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

1. Copy the application source files in to the "Files" directory, e.g.
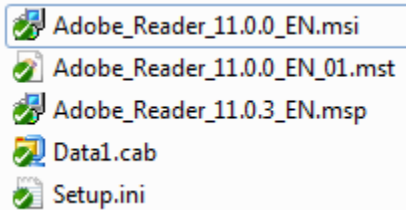
2. Customize the Deploy-Application.ps1 script using the example code below
3. Install the application by running Deploy-Application.exe
4. Uninstall the application by running Deploy-Application.exe -DeploymentType "Uninstall"

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

```
# 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

```
# 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

No actions required here

Uninstallation

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}'

Deploy the Adobe Reader installation using SCCM 2007 / SCCM 2012 package

- Copy the installation files to a network location accessible by SCCM.
- Create a new Package:
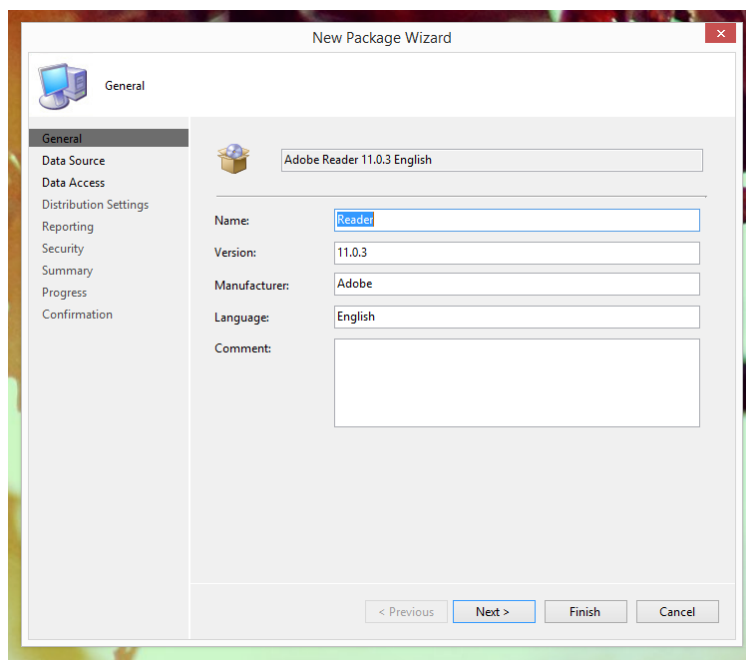
- Set the package source folder accordingly:
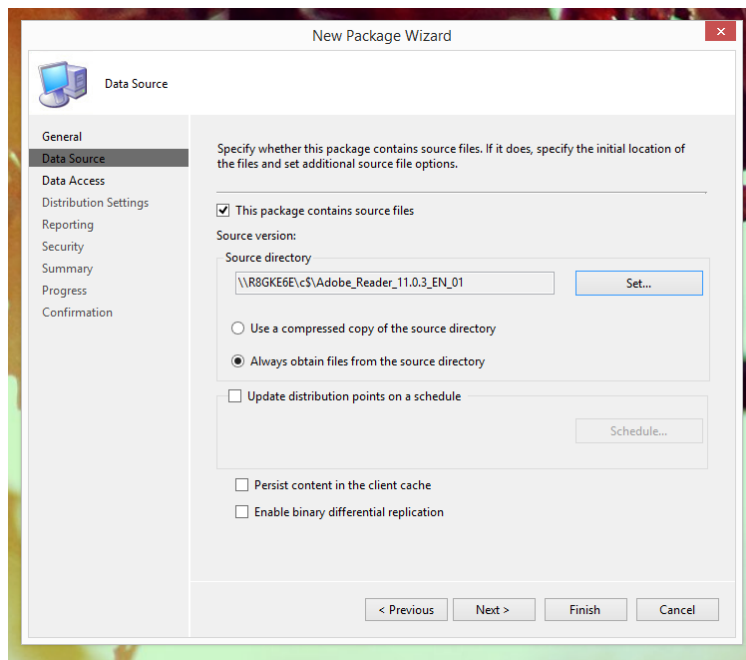
- Accept the defaults for the rest of the package (or modify according to your environment)
- Distribute the content of the package to the relevant Distribution Points
- Create a new Program for the package:
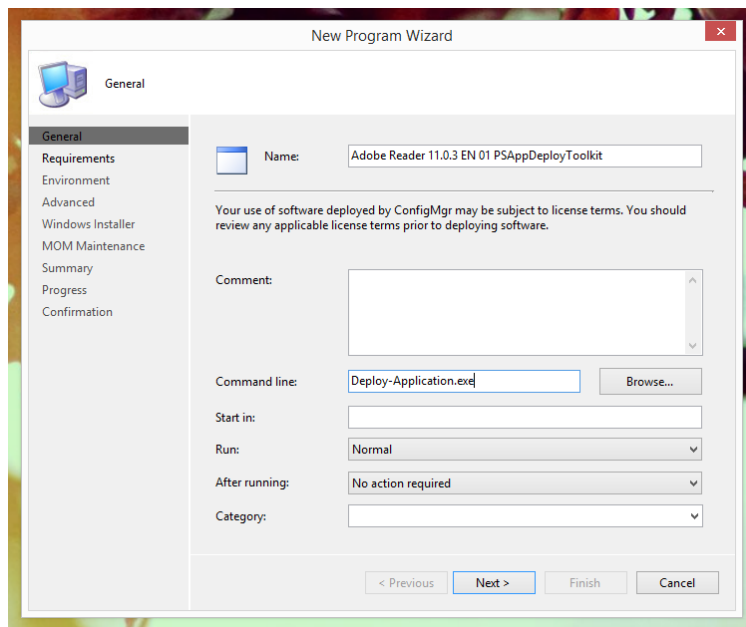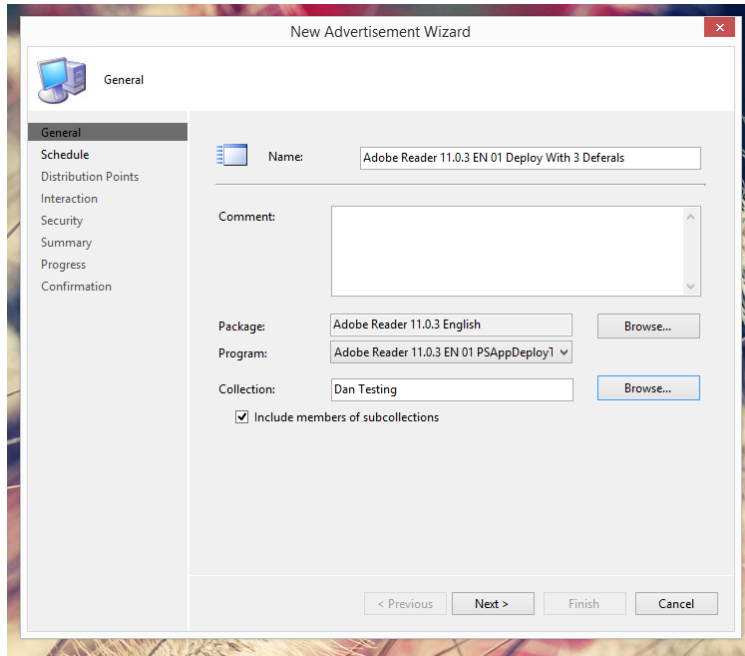
- Accept the defaults for the requirements of the program (or modify according to your environment)
- On the Environment page, ensure you use a combination of settings that allows the user to interact with the application. Failure to do so will result in the application installing silently:
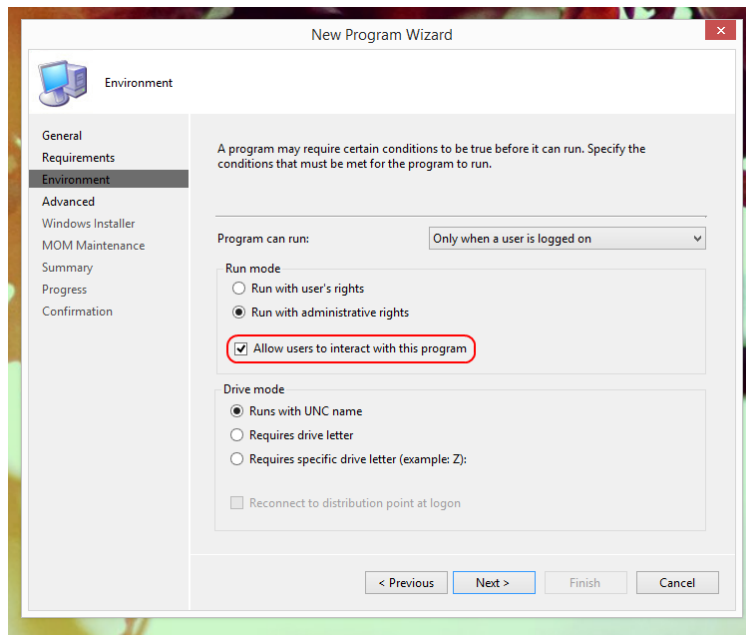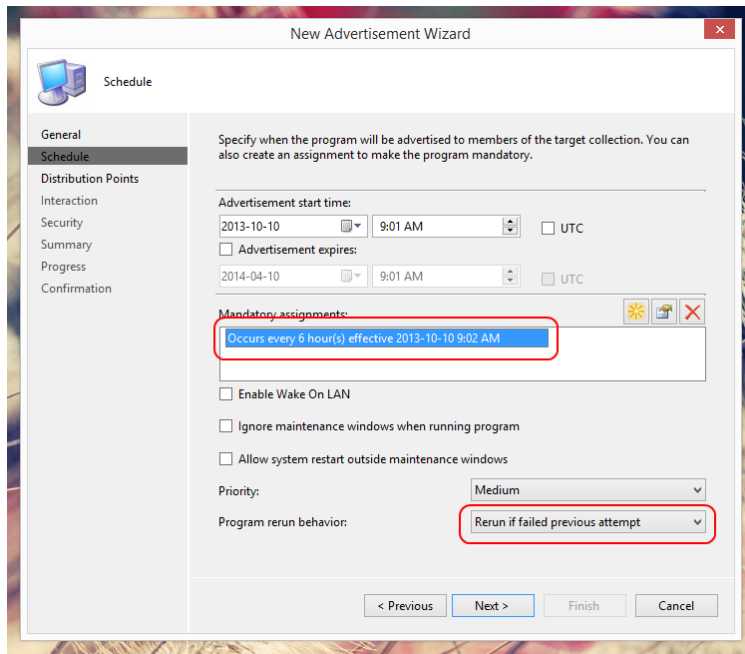
- Accept the defaults for the rest of the program (or modify according to your environment)

- Create a new Advertisement for the Package and set your target collection accordingly:

- 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:

- When prompted with the following dialog box, select Yes:
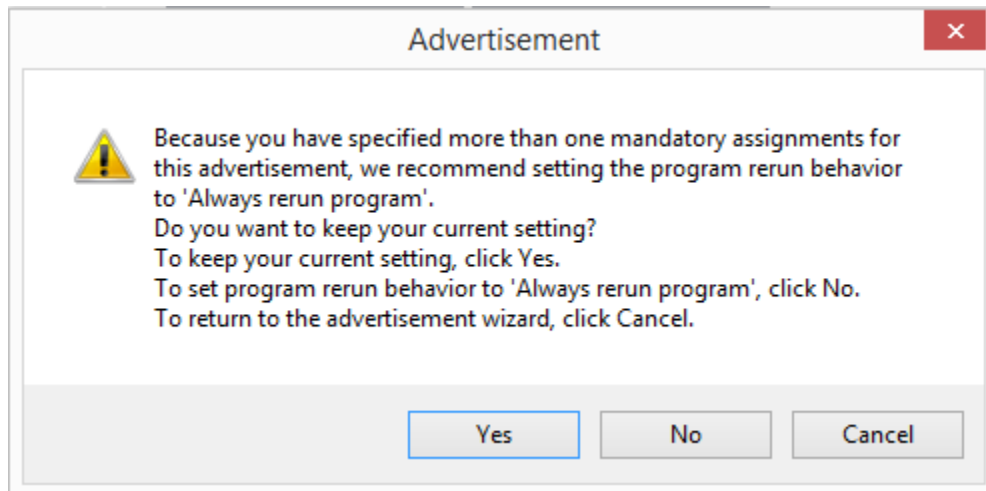
- Accept the defaults for the rest of the advertisement (or modify according to your environment). The deployment should start on your target machines shortly.

Deploy the Adobe Reader installation using SCCM 2012 Application Model

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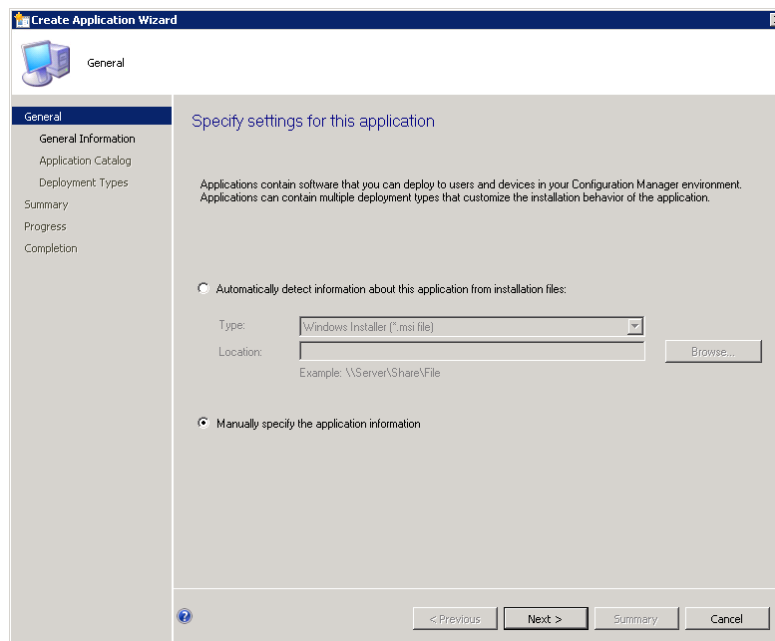
- Populate the application details accordingly:

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

- Populate the deployment type details accordingly:

- 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:

- 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:
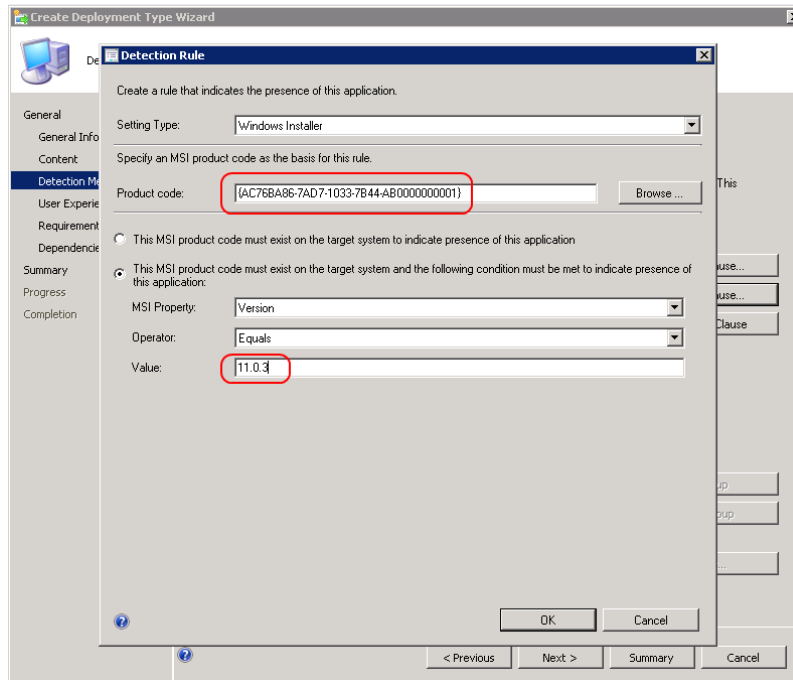
- On the User Experience page, ensure you use a combination of settings that allows the user to interact with the application. Failure to do so will result in the application installing silently:
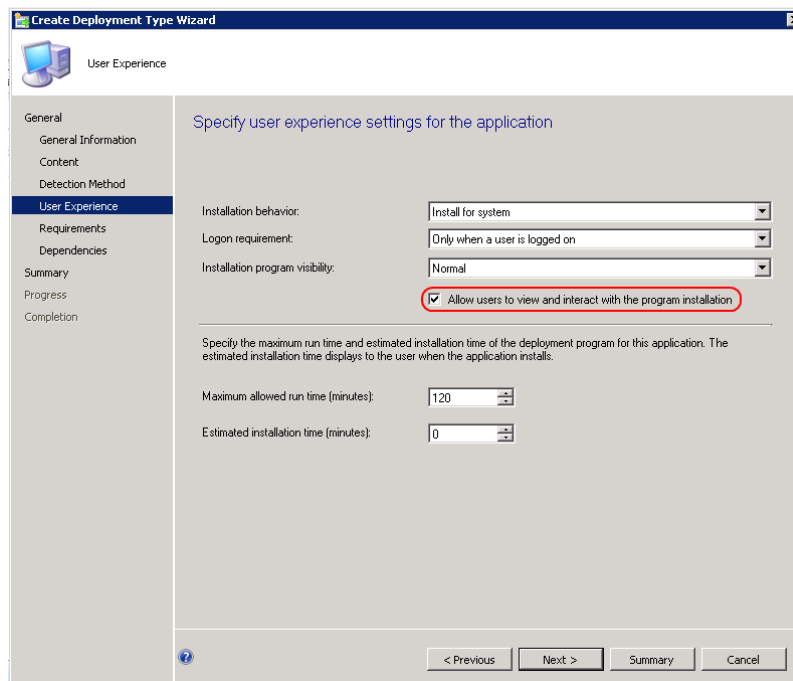
- Leave the requirements page blank (or modify according to your environment)
- Leave the software dependencies page blank (or modify according to your environment)
- Accept the defaults to create the Application
- Deploy the Application:
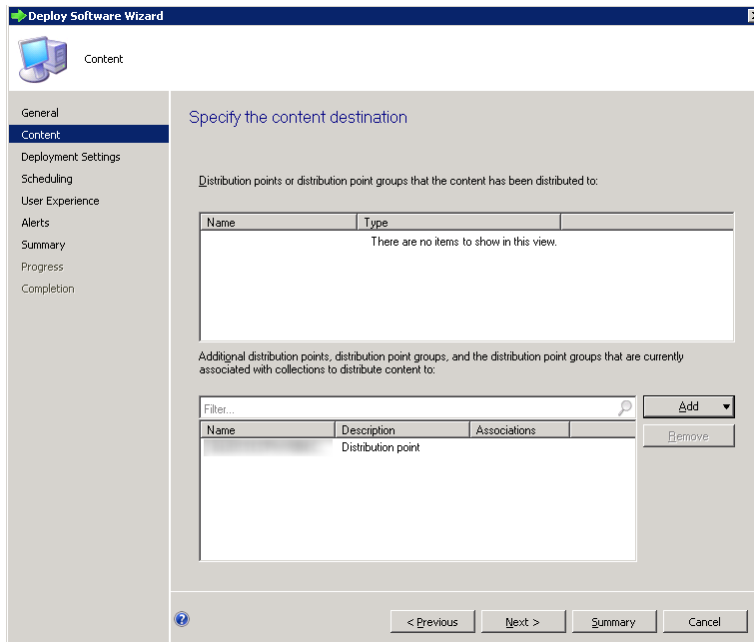
- Select the relevant Distribution Points:

- Configure deployment settings according to whether it should be a mandatory or app catalogue based deployment:
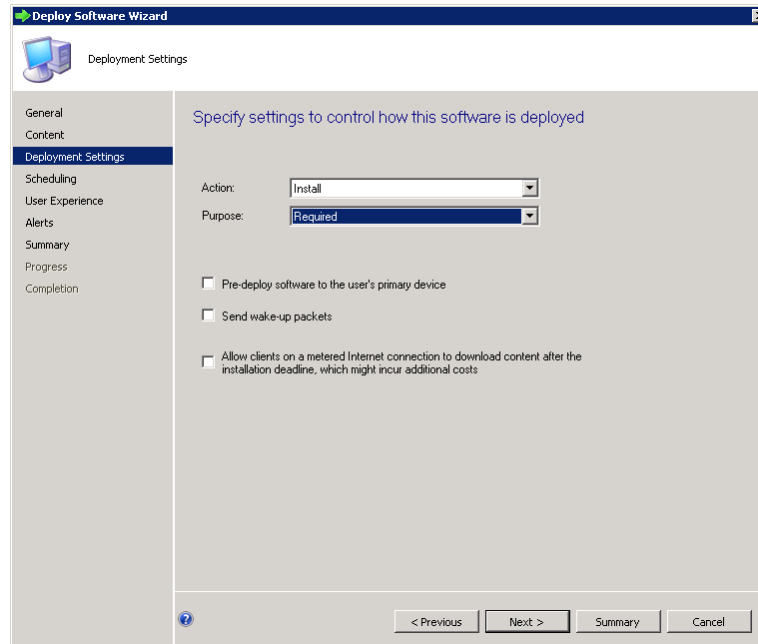
- Specify the deployment schedule:
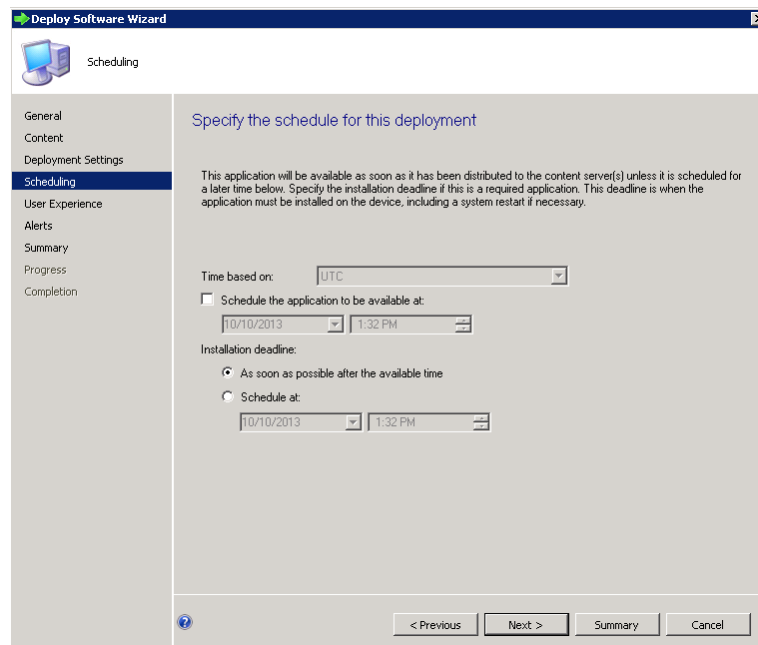
- Specify User notification settings. In order to prevent excess noise, we recommend only showing notifications for computer restarts:
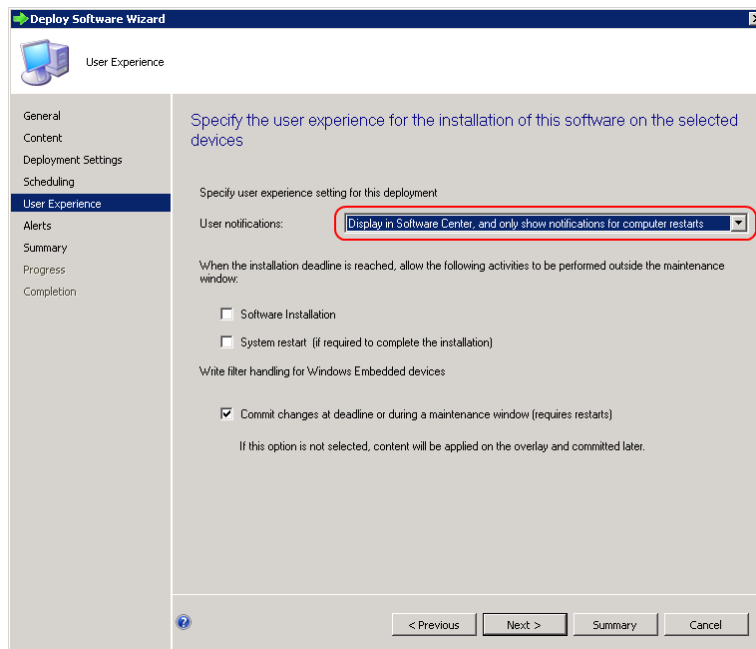
- Accept the defaults for the rest of the Deployment (or modify according to your environment)

Important Note regarding deferrals

The SCCM 2012 Application Model does not have the flexibility to schedule Mandatory Assignments on a recurring schedule like SCCM 2007 or SCCM 2012 packages do. Instead, this is determined by the frequency of Software Deployment evaluation cycle in the SCCM 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 SCCM servers and clients, and is not configurable on a per application basis:
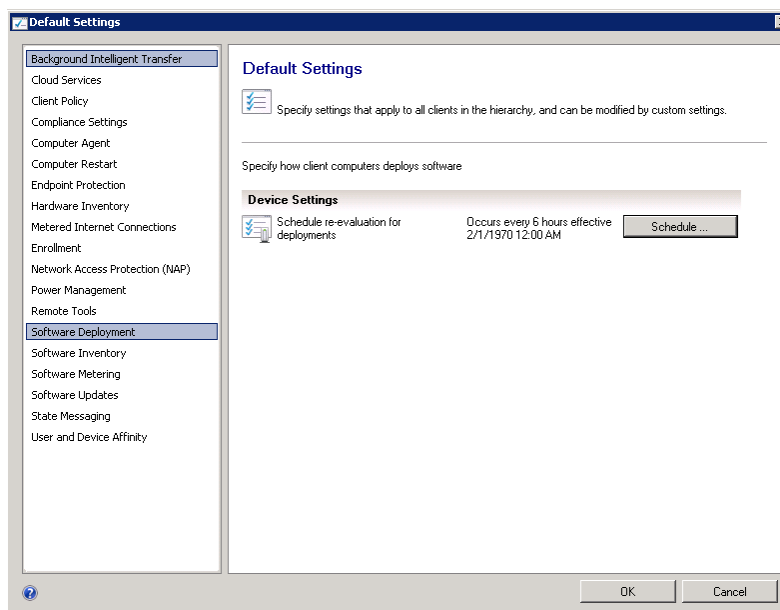
An advanced Office 2013 SP1 installation with the PowerShell App Deployment Toolkit

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:

- Deploy-Application.exe
    - o Installs Office 2010 with core products
- Deploy-Application.exe -AddInfoPath
    - o Installs Office 2010 with core products and InfoPath
- Deploy-Application.exe -AddComponentsOnly -AddInfoPath
    - o Installs InfoPath to an existing Office 2013 installation

6 Zero-Config MSI Install

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:

a) 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.
b) 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.
c) 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.

7 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)
	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% (e.g. \Users\{username})
$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:)
$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% (e.g. {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. <name>.<suffix>.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)
$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)
$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)
$appDeployToolkitDotSourceExt	Name of the optional extensions file for the toolkit (e.g. AppDeployToolkitExtensions.ps1)

ensions	
$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

$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)
$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','HKLM:SOFTWARE\Wow6432Node\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

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.

8 Toolkit Exit Codes

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

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 AppDeployToolkitExtensions.ps1

60001   An error occurred in Deploy-Application.ps1. Check your script syntax use.

60002   Error when running Execute-Process function

60003   Administrator privileges required for Execute-ProcessAsUser function

60004   Failure when loading .NET Winforms / WPF Assemblies

60005   Failure when displaying the Blocked Application dialog

60006   AllowSystemInteractionFallback option was not selected in the config XML file, so toolkit will not fall back to SYSTEM context with no interaction.

60007   Failed to export the schedule task XML file in Execute-ProcessAsUser function

60008   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.

60009   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.

60010   Deploy-Application.exe failed before PowerShell.exe process could be launched.

60011   Deploy-Application.exe failed to execute the PowerShell.exe process.

60012   A UI prompt timed out or the user opted to defer the installation.

60013   If Execute-Process function captures an exit code out of range for int32 then return this custom exit code.

9 Toolkit Functions

Convert-RegistryPath

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

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".

Parameter:

Key

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

SID

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.

Examples:

------------------------ EXAMPLE 1 ------------------------

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

------------------------ EXAMPLE 2 ------------------------

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

ConvertTo- NTAccountOrSID

Synopsis: Convert between NT Account names and their security identifiers (SIDs).

Description:

Specify either the NT Account name or the SID and get the other. Can also convert well known sid types.

Parameter:

AccountName

The Windows NT Account name specified in <domain>\<username> format.

Use fully qualified account names (e.g., <domain>\<username>) instead of isolated names (e.g., <username>) because they are unambiguous and provide better performance.

SID

The Windows NT Account SID.

WellKnownSIDName

Specify the Well Known SID name translate to the actual SID (e.g., LocalServiceSid).

To get all well-known SIDs available on system: [enum]::GetNames([Security.Principal.WellKnownSidType])

WellKnownToNTAccount

Convert the Well Known SID to an NTAccount name

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> ConvertTo-NTAccountOrSID -AccountName 'CONTOSO\User1'

------------------------ EXAMPLE 2 ------------------------

C:\PS> ConvertTo-NTAccountOrSID -SID 'S-1-5-21-1220945662-2111687655-725345543-14012660'

------------------------ EXAMPLE 3 ------------------------

C:\PS> ConvertTo-NTAccountOrSID -WellKnownSIDName 'NetworkServiceSid'

Copy-File

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

Description:

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

Parameter:

Path

Path of the file to copy.

Destination

Destination Path of the file to copy.

Recurse

Copy files in subdirectories.

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Copy-File -Path "$dirSupportFiles\MyApp.ini" -Destination "$envWindir\MyApp.ini"

------------------------ EXAMPLE 2 ------------------------

C:\PS>Copy-File -Path "$dirSupportFiles*.*" -Destination "$envTemp\tempfiles"

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

Enable-TerminalServerInstallMode

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

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

Parameter:

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Enable-TerminalServerInstallMode

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.

Description:

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.

Parameter:

Action

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

Path (Alias: FilePath)

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

Transform

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.

Patch

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.

Parameters (Alias: Arguments)

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

AddParameters

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

SecureParameters

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

LoggingOptions

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

LogName

Overrides the default log file name. The default log file name is generated from the MSI file name. If LogName does not end in .log, it will be automatically appended.

For uninstallations, the product code is resolved to the DisplayName and Version of the application.

WorkingDirectory

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

SkipMSIAlreadyInstalledCheck

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

IncludeUpdatesAndHotfixes

When checking for installed MSIs, include matches against updates and hotfixes in results.

PassThru

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

ContinueOnError

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Execute-MSI -Action 'Install' -Path 'Adobe_FlashPlayer_11.2.202.233_x64_EN.msi'

Installs an MSI

------------------------ EXAMPLE 2 ------------------------

C:\PS> 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 ------------------------

C:\PS> [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 ------------------------

C:\PS> Execute-MSI -Action 'Uninstall' -Path '{26923b43-4d38-484f-9b9e-de460746276c}'

Uninstalls an MSI using a product code

------------------------ EXAMPLE 5 ------------------------

C:\PS> Execute-MSI -Action 'Patch' -Path 'Adobe_Reader_11.0.3_EN.msp'

Installs an MSP

Execute-Process

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

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).

Parameter:

Path (Alias: FilePath)

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 (Alias: Arguments)

Arguments to be passed to the executable

SecureParameters

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

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 is not working, then check the command line options for the process being executed to see it has a silent option.

CreateNoWindow

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

WorkingDirectory

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

NoWait

Immediately continue after executing the process.

PassThru

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

WaitForMsiExec

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

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

IgnoreExitCodes

List the exit codes to ignore.

ContinueOnError

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

Examples:

----------------------- EXAMPLE 1 -----------------------

C:\PS> 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 -----------------------

C:\PS> Execute-Process -Path "$dirFiles\Bin\setup.exe" -Parameters '/S' -WindowStyle 'Hidden'

----------------------- EXAMPLE 3 -----------------------

C:\PS> Execute-Process -Path 'setup.exe' -Parameters '/S' -IgnoreExitCodes '1,2'

----------------------- EXAMPLE 4 -----------------------

C:\PS> 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 -----------------------

```
C:\PS> 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.

Description:

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

Parameter:

UserName

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

Path to the file being executed.

Parameters

Arguments to be passed to the file being executed.

SecureParameters

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

RunLevel

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

- HighestAvailable: 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

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

PassThru

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

ContinueOnError

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> 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 ------------------------

C:\PS> 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.

Description:

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

Performs cleanup actions such as closing down dialogs and unblocking blocked applications.

Displays a balloon tip notification to indicate the setup is complete and whether it was a success or a failure.

Determines what exit code to pass to the parent process depending on the options specified in the deployment script, e.g.

If $AllowRebootPassThru is set to False, it will suppress any "3010" exit codes detected during the installation and instead pass the "0" exit code.

Parameter:

ExitCode

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Exit-Script -ExitCode 0

------------------------ EXAMPLE 2 ------------------------

C:\PS>Exit-Script -ExitCode 1618

Disable-TerminalServerInstallMode

Synopsis: Changes to user execute mode for Remote Desktop Session Host/Citrix servers

Description:

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

Parameter:

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Disable-TerminalServerInstallMode

Get-FileVersion

Synopsis: Gets the version of the specified file

Description:

Gets the version of the specified file

Parameter:

File

Path of the file

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Get-FileVersion -File "$envProgramFilesX86\Adobe\Reader 11.0\Reader\AcroRd32.exe"

Get-HardwarePlatform

Synopsis: Retrieves information about the hardware platform (physical or virtual)

Description:

Retrieves information about the hardware platform (physical or virtual)

Parameter:

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Get-HardwarePlatform

Get-FreeDiskSpace

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

Description:

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

Parameter:

Drive

Drive to check free disk space on

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-FreeDiskSpace -Drive "C:"

Get-IniValue

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

Description:

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

Parameter:

FilePath

Path to the INI file

Section

Section within the INI file

Key

Key within the section of the INI file

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-IniValue -FilePath "$envProgramFilesX86\IBM\Notes\notes.ini" -Section 'Notes' -Key 'KeyFileName'

Get-InstalledApplication

Synopsis: Retrieves information about installed applications.

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.

Parameter:

Name

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

Exact

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

WildCard

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

RegEx

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

ProductCode

The product code of the application you want to retrieve information on.

IncludeUpdatesAndHotfixes

Include matches against updates and hotfixes in results.

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-InstalledApplication -Name 'Adobe Flash'

------------------------ EXAMPLE 2 ------------------------

C:\PS> Get-InstalledApplication -ProductCode '{1AD147D0-BE0E-3D6C-AC11-64F6DC4163F1}'

Get- LoggedOnUser

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

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

Parameter:

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 listening 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 is 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.

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-LoggedOnUser

Get-PendingReboot

Synopsis: Get the pending reboot status on a local computer.

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) Pending File Rename Operations (XP, Windows 2003 / 2008)

Parameter:

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-PendingReboot

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

*Notes: ErrorMsg only contains something if an error occurred

------------------------ EXAMPLE 2 ------------------------

C:\PS> (Get-PendingReboot).IsSystemRebootPending

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

Get-RegistryKey

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

Description:

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

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

Parameter:

Key

Path of the registry key

Value

Value to retrieve (optional)

SID

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

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

DoNotExpandEnvironmentNames

Return unexpanded REG_EXPAND_SZ values. Default is: $false.

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

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

------------------------ EXAMPLE 2 ------------------------

C:\PS> Get-RegistryKey -Key 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Image File Execution Options\iexplore.exe'

------------------------ EXAMPLE 3 ------------------------

C:\PS> 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-ScheduledTask

Synopsis: Retrieve all details for scheduled tasks on the local computer.

Description:

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

Parameter:

TaskName

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

ContinueOnError

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-ScheduledTask

To display a list of all scheduled task properties.

------------------------ EXAMPLE 2 ------------------------

C:\PS> Get-ScheduledTask | Out-GridView

To display a grid view of all scheduled task properties.

------------------------ EXAMPLE 3 ------------------------

C:\PS> Get-ScheduledTask | Select-Object -Property TaskName

To display a list of all scheduled task names.

Get- ServiceStartMode

Synopsis: Get the service startup mode.

Description:

Get the service startup mode.

Parameter:

Name

Specify the name of the service.

ComputerName

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

ContinueOnError

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-ServiceStartMode -Name 'wuauserv'

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).

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 may be empty for some user profiles but the SID and ProfilePath properties will always be populated.

Parameter:

ExcludeNTAccount

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

ExcludeSystemProfiles

Exclude system profiles: SYSTEM, LOCAL SERVICE, NETWORK SERVICE. Default is: $true.

ExcludeDefaultUser

Exclude the Default User. Default is: $false.

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-UserProfiles

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

------------------------ EXAMPLE 2 ------------------------

C:\PS> Get-UserProfiles -ExcludeNTAccount 'CONTOSO\Robot','CONTOSO\ntadmin'

------------------------ EXAMPLE 3 ------------------------

C:\PS> [string[]]$ProfilePaths = Get-UserProfiles | Select-Object -ExpandProperty 'ProfilePath'

Returns the user profile path for each user on the system. This information can then be used to make modifications under the user profile on the filesystem.

Get-WindowTitle

Synopsis: Search for an open window title and return details about the window.

Description:

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

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.

Parameter:

WindowTitle

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

GetAllWindowTitles

Get titles for all open windows on the system.

DisableFunctionLogging

Disables logging messages to the script log file.

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Get-WindowTitle -WindowTitle 'Microsoft Word'

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

------------------------ EXAMPLE 2 ------------------------

C:\PS> Get-WindowTitle -GetAllWindowTitles

Gets details for all windows with a title.

------------------------ EXAMPLE 3 ------------------------

C:\PS> Get-WindowTitle -GetAllWindowTitles | Where-Object { $_.ParentProcess -eq 'WINWORD' }

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

Install-MSUpdates

Synopsis: Installs all Microsoft Updates in a given directory

Description:

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

Parameter:

Directory

Directory containing the updates

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Install-MSUpdates -Directory "$dirFiles\MSUpdates"

Install-SCCMSoftwareUpdates

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

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.

Parameter:

SoftwareUpdatesScanWaitInSeconds

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

WaitForPendingUpdatesTimeout

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

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Install-SCCMSoftwareUpdates

Invoke-HKCURegistrySettingsForAllUsers

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

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.

Parameter:

RegistrySettings

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

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

Examples:

------------------------- EXAMPLE 1 -------------------------

C:\PS>[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 (Alias: Register-DLL, Unregister-DLL)

Synopsis: Register or unregister a DLL file.

Description:

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

Parameter:

FilePath

Path to the DLL file

DLLAction

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

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

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

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

------------------------ EXAMPLE 2 ------------------------

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

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

------------------------ EXAMPLE 3 ------------------------

C:\PS> 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.

Description:

Triggers SCCM to invoke the requested schedule task id.

Parameter:

ScheduleID

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

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Invoke-SCCMTask 'SoftwareUpdatesScan'

------------------------ EXAMPLE 2 ------------------------

C:\PS>Invoke-SCCMTask

New-Folder

Synopsis: Create a new folder.

Description:

Create a new folder if it does not exist.

Parameter:

Path

Path to the new folder which should be created.

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> New-Folder -Path "$envWinDir\System32"

New-MsiTransform

Synopsis: Create a transform file for an MSI database.

Description:

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

Parameter:

MsiPath

Specify the path to an MSI file.

ApplyTransformPath

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

NewTransformPath

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 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

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

Examples:

------------------------- EXAMPLE 1 -------------------------

C:\PS> [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

Description:

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

Parameter:

Path

Path to save the shortcut

TargetPath

Target path or URL that the shortcut launches

Arguments

Arguments to be passed to the target path

IconLocation

Location of the icon used for the shortcut

IconIndex

Executables, DLLs, ICO files with multiple icons need the icon index to be specified

Description

Description of the shortcut

WorkingDirectory

Working Directory to be used for the target path

WindowStyle

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

> *RunAsAdmin*

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

> *ContinueOnError*

> Continue if an error is encountered

Examples:

> ------------------------ EXAMPLE 1 ------------------------

> C:\PS>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"

Refresh-Desktop

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

Description:

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

Parameter:

> *ContinueOnError*

> Continue if an error is encountered

Examples:

> ------------------------ EXAMPLE 1 ------------------------

> C:\PS>Refresh-Desktop

Refresh-SessionEnvironmentVariables

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

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.

Parameter:

> *LoadLoggedOnUserEnvironmentVariables*

> 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

Continue if an error is encountered

Examples:

------------------------- EXAMPLE 1 -------------------------

C:\PS>Refresh-SessionEnvironmentVariables

Remove-File

Synopsis: Remove a file or all files recursively in a given path.

Description:

Remove a file or all files recursively in a given path.

Parameter:

Path

Path of the file to remove.

LiteralPath

Path of the file to remove. Wildcard characters are ignored.

Recurse

Optionally, remove all files recursively in a directory.

ContinueOnError

Continue if an error is encountered

Examples:

------------------------- EXAMPLE 1 -------------------------

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

------------------------- EXAMPLE 2 -------------------------

C:\PS>Remove-File -Path 'C:\Windows\Downloaded Program Files' -Recurse

Remove-Folder

Synopsis: Remove folder and files if they exist.

Description:

Remove folder and all files recursively in a given path.

Parameter:

Path

Path to the folder which should be removed.

ContinueOnError

Continue if an error is encountered

Examples:

------------------------- EXAMPLE 1 -------------------------

C:\PS> Remove-Folder -Path "$envWinDir\Downloaded Program Files"

Remove-MSIApplications

Synopsis: Removes all MSI applications matching the specified application name

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".

Parameter:

Name

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

Exact

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

WildCard

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

Parameters (Alias: Arguments)

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

AddParameters

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

FilterApplication

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

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

When checking for installed MSIs, include matches against updates and hotfixes in results.

LoggingOptions

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

LogName

Overrides the default log file name. The default log file name is generated from the MSI file name. If LogName does not end in .log, it will be automatically appended. For uninstallations, by default the product code is resolved to the DisplayName and Version of the application.

PassThru

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

ContinueOnError

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Remove-MSIApplications -Name 'Adobe Flash'

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

------------------------ EXAMPLE 2 ------------------------

C:\PS> Remove-MSIApplications -Name 'Adobe'

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

------------------------ EXAMPLE 3 ------------------------

C:\PS> 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 ------------------------

C:\PS> Remove-MSIApplications -Name 'Java 8 Update' -FilterApplication @(,,@('Publisher', 'Oracle Corporation', 'Exact')) -ExcludeFromUninstall @(,,@('DisplayName', 'Java 8 Update 45', 'RegEx'))

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 array in an array of arrays, the array must be preceded by two commas as in this example.

------------------------ EXAMPLE 5 ------------------------

C:\PS> Remove-MSIApplications -Name 'Java 8 Update' -ExcludeFromUninstall @(,,@('DisplayName', 'Java 8 Update 45', 'RegEx'))

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 array in an array of arrays, the array must be preceded by two commas as in this example.

------------------------- EXAMPLE 6 -------------------------

C:\PS> Remove-MSIApplications -Name 'Java 8 Update' -ExcludeFromUninstall @(

      @('Is64BitApplication', $true, 'Exact'),

      @('DisplayName', 'Java 8 Update 45', 'Exact'),

      @('DisplayName', 'Java 8 Update 4*', 'WildCard'),

      @('DisplayName', 'Java 8 Update 45', 'RegEx')

)

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.

Notes :

More reading on how to create arrays if having trouble with -ExcludeFromUninstall parameter:
http://blogs.msdn.com/b/powershell/archive/2007/01/23/array-literals-in-powershell.aspx

Remove-RegistryKey

Synopsis: Deletes the specified registry key or value

Description:

Deletes the specified registry key or value

Parameter:

Key

Path of the registry key to delete

Name

Name of the registry key value to delete

Recurse

Delete registry key recursively.

SID

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

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Remove-RegistryKey -Key 'HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce'

------------------------ EXAMPLE 2 ------------------------

C:\PS> Remove-RegistryKey -Key 'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Run' -Name 'RunAppInstall'

Resolve-Error

Synopsis: Enumerate error record details.

Description:

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

Parameter:

ErrorRecord

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

The list of properties to display from the error record. Use "*" to display all properties. Default list of error properties is: Message, FullyQualifiedErrorId, ScriptStackTrace, PositionMessage, InnerException

GetErrorRecord

Get error record details as represented by $_.

GetErrorInvocation

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

GetErrorException

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

GetErrorInnerException

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Resolve-Error

------------------------ EXAMPLE 2 ------------------------

C:\PS> Resolve-Error -Property *

------------------------ EXAMPLE 3 ------------------------

C:\PS> Resolve-Error -Property InnerException

------------------------ EXAMPLE 4 ------------------------

C:\PS> Resolve-Error –GetErrorInvocation $false

Send-Keys

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

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.

Parameter:

WindowTitle

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

GetAllWindowTitles

Get titles for all open windows on the system.

WindowHandle

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

Keys

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

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

Examples:

----------------------- EXAMPLE 1 -----------------------

C:\PS> Send-Keys -WindowTitle 'foobar - Notepad' -Key 'Hello world'

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

----------------------- EXAMPLE 2 -----------------------

C:\PS> 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 -----------------------

C:\PS> 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.

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).

Parameter:

StubExePath

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

Arguments to pass to the file being executed.

Description

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

Key

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

Version

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

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

PurgeActiveSetupKey

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

DisableActiveSetup

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

ContinueOnError

Continue if an error is encountered.

Examples:

------------------------- EXAMPLE 1 -------------------------

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

------------------------- EXAMPLE 2 -------------------------

C:\PS> 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 2 -------------------------

C:\PS> 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.

Description:

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

Parameter:

FilePath

Path to the INI file

Section

Section within the INI file

Key

Key within the section of the INI file

Value

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

ContinueOnError

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> 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.

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.

Parameter:

Action

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

FilePath

Path to the shortcut file to be pinned or unpinned

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Set-PinnedApplication -Action 'PintoStartMenu' -FilePath "$envProgramFilesX86\IBM\Lotus\Notes\notes.exe"

------------------------ EXAMPLE 2 ------------------------

C:\PS> 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.

Description:

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

Parameter:

Key

The registry key path

Name

The value name

Value

The value data

Type

The type of registry value to create or set. Options: 'Binary', 'DWord', 'ExpandString', 'MultiString', 'None', 'QWord', 'String', 'Unknown'. Default: String.

SID

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

Continue if an error is encountered

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Set-RegistryKey -Key $blockedAppPath -Name 'Debugger' -Value $blockedAppDebuggerValue

------------------------ EXAMPLE 2 ------------------------

C:\PS> Set-RegistryKey -Key
'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce' -Name 'Debugger' -
Value $blockedAppDebuggerValue -Type String

------------------------ EXAMPLE 3 ------------------------

C:\PS> 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'

Set-ServiceStartMode

Synopsis: Set the service startup mode.

Description:

Set the service startup mode.

Parameter:

Name

Specify the name of the service.

ComputerName

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

StartMode

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

ContinueOnError

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

Examples:

------------------------- EXAMPLE 1 -------------------------

C:\PS> Set-ServiceStartMode -Name 'wuauserv' -StartMode 'Automatic (Delayed Start)'

Show-BalloonTip

Synopsis: Displays a balloon tip notification in the system tray

Description:

Displays a balloon tip notification in the system tray

Parameter:

BalloonTipText

Text of the balloon tip

BalloonTipTitle

Title of the balloon tip

BalloonTipIcon

Toolkit Functions | PowerShell App Deployment Toolkit

Time in milliseconds to display the balloon tip. Default: 500.

BalloonTipTime

Time in milliseconds to display the balloon tip [Default 500]

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Show-BalloonTip -BalloonTipText 'Installation Started' -BalloonTipTitle 'Application Name'

------------------------ EXAMPLE 2 ------------------------

C:\PS> 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.

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.

Parameter:

Text

Text in the message dialog box

Title

Title of the message dialog box

Buttons

Buttons to be included on the dialog box. Options: OK, OKCancel, AbortRetryIgnore, YesNoCancel, YesNo, RetryCancel, CancelTryAgainContinue. Default: OK.

DefaultButton

The Default button that is selected. Options: First, Second, Third. Default: First.

Icon

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

Timeout

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

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Show-DialogBox -Title 'Installed Complete' -Text 'Installation has completed. Please click OK and restart your computer.' -Icon 'Information'

------------------------ EXAMPLE 2 ------------------------

C:\PS> 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

Show-InstallationProgress

Synopsis: Displays a progress dialog in a separate thread with an updatable custom message.

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).

Parameter:

StatusMessage

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

WindowLocation

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

TopMost

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Show-InstallationProgress

Uses the default status message from the XML configuration file.

------------------------ EXAMPLE 2 ------------------------

C:\PS> Show-InstallationProgress -StatusMessage 'Installation in Progress...'

------------------------ EXAMPLE 3 ------------------------

C:\PS> Show-InstallationProgress -StatusMessage "Installation in Progress...`nThe installation may take 20 minutes to complete."

----------------------- EXAMPLE 4 -----------------------

C:\PS> 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.

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.

Parameter:

Title

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

Message

Message text to be included in the prompt

MessageAlignment

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

ButtonLeftText

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

ButtonRightText

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

ButtonMiddleText

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

Icon

Show a system icon in the prompt. Options: Application, Asterisk, Error, Exclamation, Hand, Information, None, Question, Shield, Warning, WinLogo. Default: None.

NoWait

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

PersistPrompt

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

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

Timeout

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

ExitOnTimeout

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Show-InstallationPrompt -Message 'Do you want to proceed with the installation?' -ButtonRightText 'Yes' -ButtonLeftText 'No'

------------------------ EXAMPLE 2 ------------------------

C:\PS> Show-InstallationPrompt -Title 'Funny Prompt' -Message 'How are you feeling today?' -ButtonRightText 'Good' -ButtonLeftText 'Bad' -ButtonMiddleText 'Indifferent'

------------------------ EXAMPLE 3 ------------------------

C:\PS> 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.

Description:

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

Parameter:

CountdownSeconds

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

CountdownNoHideSeconds

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

NoCountdown

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.

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Show-InstallationRestartPrompt -Countdownseconds 600 -CountdownNoHideSeconds 60

------------------------ EXAMPLE 2 ------------------------

C:\PS> 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.

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).

Parameter:

CloseApps

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

Stop processes without prompting the user.

CloseAppsCountdown

Option to provide a countdown in seconds until the specified applications are automatically closed. This only takes effect if deferral is now allowed or has expired.

ForceCloseAppsCountdown

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

PromptToSave

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

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!

BlockExecution

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

AllowDefer

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

AllowDeferCloseApps

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

Specify the number of times the installation can be deferred

DeferDays

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

DeferDeadline

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

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

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, e.g. "2013-08-22 11:51:52Z"

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

CheckDiskSpace

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

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

MinimizeWindows

Specifies whether to minimize other windows when displaying prompt [Default is true]

TopMost

Specifies whether the window is the topmost window [Default is true]

ForceCountdown

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

CustomText

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Show-InstallationWelcome -CloseApps 'iexplore,winword,excel'

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

------------------------ EXAMPLE 2 ------------------------

C:\PS> Show-InstallationWelcome -CloseApps 'winword,excel' -Silent

Close Word and Excel without prompting the user.

------------------------ EXAMPLE 3 ------------------------

C:\PS> 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 ------------------------

C:\PS> 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 ------------------------

C:\PS> 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 ------------------------

C:\PS> Show-InstallationWelcome -AllowDefer -DeferDeadline '25/08/2013'

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

------------------------ EXAMPLE 7 ------------------------

C:\PS> 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.

Description:

Start Windows service and its dependencies.

Parameter:

Name

Specify the name of the service.

ComputerName

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

SkipServiceExistsTest

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

SkipDependentServices

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

PendingStatusWait

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

PassThru

Return the System.ServiceProcess.ServiceController service object.

ContinueOnError

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Start-ServiceAndDependencies -Name 'wuauserv'

Stop-ServiceAndDependencies

Synopsis: Stop Windows service and its dependencies.

Description:

    Stop Windows service and its dependencies.

Parameter:

    Name

    Specify the name of the service.

    ComputerName

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

    SkipServiceExistsTest

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

    SkipDependentServices

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

    PendingStatusWait

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

    PassThru

    Return the System.ServiceProcess.ServiceController service object.

    ContinueOnError

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

Examples:

    ------------------------- EXAMPLE 1 -------------------------

    C:\PS>Stop-ServiceAndDependencies -Name 'wuauserv'

Test-Battery

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

Description:

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

Parameter:

    PassThru

    Outputs a hashtable containing the following properties:

    IsLaptop, IsUsingACPower, ACPowerLineStatus, BatteryChargeStatus, BatteryLifePercent, BatteryLifeRemaining, BatteryFullLifetime

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Test-Battery

------------------------ EXAMPLE 2 ------------------------

C:\PS>(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

Description:

Test whether a Microsoft Windows update is installed

Parameter:

KBNumber

KBNumber of the update.

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Test-MSUpdates -KBNumber 'KB2549864'

Test-NetworkConnection

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

Description:

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

Parameter:

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Test-NetworkConnection

Test-PowerPoint

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

Description:

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

Parameter:

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Test-PowerPoint

Test-RegistryValue

Synopsis: Test if a registry value exists.

Description:

Checks a registry key path to see if it has a value with a given name. Can correctly handle cases where a value simply has an empty or null value.

Parameter:

Key

Path of the registry key.

Value

Specify the registry key value to check the existence of.

SID

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.

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Test-RegistryValue -Key 'HKLM:SYSTEM\CurrentControlSet\Control\Session Manager' -Value 'PendingFileRenameOperations'

Test-ServiceExists

Synopsis: Check to see if a service exists.

Description:

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

Parameter:

Name

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

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

PassThru

Return the WMI service object.

ContinueOnError

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

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS> Test-ServiceExists -Name 'wuauserv'

------------------------ EXAMPLE 2 ------------------------

C:\PS> 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-GroupPolicy

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

Description:

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

Parameter:

Examples:

------------------------ EXAMPLE 1 ------------------------

C:\PS>Update-GroupPolicy

Write-Log

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

Description:

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

Parameter:

Message

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

Severity

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

The source of the message being logged.

ScriptSection

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

LogType

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

LogFileDirectory

Set the directory where the log file will be saved.

LogFileName

Set the name of the log file.

MaxLogFileSizeMB

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

WriteHost

Write the log message to the console.

ContinueOnError

Suppress writing log message to console on failure to write message to log file.

PassThru

Passes the text back to the PowerShell pipeline

DebugMessage

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

LogDebugMessage

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

Examples:

------------------------- EXAMPLE 1 -------------------------

C:\PS>Write-Log -Message "Installing patch MS15-031" -Source 'Add-Patch' -LogType 'CMTrace'

------------------------- EXAMPLE 2 -------------------------

C:\PS>Write-Log -Message "Script is running on Windows 8" -Source 'Test-ValidOS' -LogType 'Legacy'