**C:\Scripts\PSAppDeployToolkit\Toolkit\AppDeployToolkit\AppDeployToolkitBanner.png**

Administrator Guide

<http://psappdeploytoolkit.codeplex.com>

Document Version **3.5**

Contents

[Administrator Guide 1](#_Toc395728878)

[2 Overview 5](#_Toc395728879)

[Introduction 5](#_Toc395728880)

[Features 5](#_Toc395728881)

[System Requirements and Support 6](#_Toc395728882)

[Licensing 6](#_Toc395728883)

[3 Toolkit Functionality 6](#_Toc395728884)

[User Interface 6](#_Toc395728885)

[Functions/Logic 7](#_Toc395728886)

[Integration with SCCM 8](#_Toc395728887)

[Help Console 8](#_Toc395728888)

[4 Toolkit Components 8](#_Toc395728889)

[Toolkit File Structure 8](#_Toc395728890)

[Files 8](#_Toc395728891)

[Directories 9](#_Toc395728892)

[Toolkit User Interface 10](#_Toc395728893)

[Installation Progress 10](#_Toc395728894)

[Installation Welcome Prompt 11](#_Toc395728895)

[Block Application Execution 13](#_Toc395728896)

[Disk Space Requirements 13](#_Toc395728897)

[Custom Installation Prompt 14](#_Toc395728898)

[Installation Restart Prompt 15](#_Toc395728899)

[Balloon tip notifications 15](#_Toc395728900)

[Custom Dialog box 16](#_Toc395728901)

[Logging 17](#_Toc395728902)

[5 Toolkit Usage 17](#_Toc395728903)

[Overview 17](#_Toc395728904)

[Launching the Toolkit 17](#_Toc395728905)

[Overview 18](#_Toc395728906)

[Toolkit Parameters 18](#_Toc395728907)

[Customizing the Toolkit 19](#_Toc395728908)

[Example Deployments 19](#_Toc395728909)

[Building an Adobe Reader installation with the PowerShell App Deployment 19](#_Toc395728910)

[Deploy the Adobe Reader installation using SCCM 2007 / SCCM 2012 package 21](#_Toc395728911)

[Deploy the Adobe Reader installation using SCCM 2012 Application Model 25](#_Toc395728912)

[Important Note regarding deferrals 31](#_Toc395728913)

[An advanced Office 2013 SP1 installation with the PowerShell App Deployment 32](#_Toc395728914)

[6 Toolkit Variables 33](#_Toc395728915)

[7 Toolkit Functions 34](#_Toc395728916)

[Convert-RegistryPath 34](#_Toc395728917)

[Copy-File 35](#_Toc395728918)

[Enable-TerminalServerInstallMode 35](#_Toc395728919)

[Execute-MSI 35](#_Toc395728920)

[Execute-Process 37](#_Toc395728921)

[Exit-Script 38](#_Toc395728922)

[Disable-TerminalServerInstallMode 39](#_Toc395728923)

[Get-FileVersion 39](#_Toc395728924)

[Get-HardwarePlatform 40](#_Toc395728925)

[Get-IniValue 40](#_Toc395728926)

[Get-FreeDiskSpace 41](#_Toc395728927)

[Get-HardwarePlatform 41](#_Toc395728928)

[Get-InstalledApplication 41](#_Toc395728929)

[Get-RegistryKey 42](#_Toc395728930)

[Get-ScheduledTask 43](#_Toc395728931)

[Install-MSUpdates 43](#_Toc395728932)

[Install-SCCMSoftwareUpdates 43](#_Toc395728933)

[Invoke-SCCMTask 44](#_Toc395728934)

[New-Folder 44](#_Toc395728935)

[New-Shortcut 44](#_Toc395728936)

[Refresh-Desktop 45](#_Toc395728937)

[Register-DLL 46](#_Toc395728938)

[Remove-File 46](#_Toc395728939)

[Remove-Folder 46](#_Toc395728940)

[Remove-MSIApplications 47](#_Toc395728941)

[Remove-RegistryKey 47](#_Toc395728942)

[Send-Keys 48](#_Toc395728943)

[Set-IniValue 49](#_Toc395728944)

[Set-PinnedApplication 50](#_Toc395728945)

[Set-RegistryKey 50](#_Toc395728946)

[Show-BalloonTip 51](#_Toc395728947)

[Show-DialogBox 52](#_Toc395728948)

[Show-InstallationProgress 53](#_Toc395728949)

[Show-InstallationPrompt 54](#_Toc395728950)

[Show-InstallationRestartPrompt 55](#_Toc395728951)

[Show-InstallationWelcome 56](#_Toc395728952)

[Test-Battery 59](#_Toc395728953)

[Test-MSUpdates 59](#_Toc395728954)

[Test-NetworkConnection 60](#_Toc395728955)

[Test-PowerPoint 60](#_Toc395728956)

[Unregister-DLL 60](#_Toc395728957)

[Update-GroupPolicy 61](#_Toc395728958)

[Write-Log 61](#_Toc395728959)

# 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>

# 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 in several languages and more can easily be 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
* Test network connectivity
* Test power connectivity
* Check whether a PowerPoint slideshow is running

## 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.

# 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 AppDeployToolkit.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 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 easily be 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**

# 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 –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 /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 “Custom-Script.ps1”**

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

### Toolkit Parameters

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

**-DeploymentType** “Install” | “Uninstall” (default is install)

Specifies whether to install or uninstall the application.

**-DeployMode** “Interactive” | “Silent” | “NonInteractive” (default is interactive)

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

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

NonInteractive = Very silent, i.e. no blocking apps. Noninteractive mode is automatically set if an SCCM task sequence or session 0 is detected.

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

Specifies 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 Destkop Session Host/Citrix servers

## 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 and log file location.

**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.

## Example Deployments

### Building an Adobe Reader installation with the PowerShell App Deployment

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, ie. 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.



1. Customize the Deploy-Application.ps1 script using the example code below
2. Install the application by running Deploy-Application.exe
3. 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 "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 Install” and “Deploy-Application.exe 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 catalog 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

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
  + Installs Office 2010 with core products
* Deploy-Application.exe -addInfoPath
  + Installs Office 2010 with core products and InfoPath
* Deploy-Application.exe –addComponentsOnly –addInfoPath
  + Installs InfoPath to an existing Office 2013 installation

# Toolkit Variables

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

**$dirFiles** Files directory of the toolkit

**$dirSupportFiles** Supportfiles directory of the toolkit

**$currentDate** Current date

**$currentTime** Current time

**$currentLanguage** Current language (e.g. EN, FR, DE, JA etc)

**$is64Bit** Is the OS 64-Bit (true / false)

**$Is64BitProcess** Is PowerShell running 64-bit (true / false)

**$isServerOS** Is a Server OS (true / false)

**$envAllUsersProfile** %ALLUSERSPROFILE%

**$envAppData** %APPDATA%

**$envArchitecture** %PROCESSOR\_ARCHITECTURE%

**$envCommonProgramFiles** %COMMONPROGRAMFILES%

**$envCommonProgramFilesX86** %COMMONPROGRAMFILES(X86)%

**$envComputerName** %COMPUTERNAME%

**$envHomeDrive** %HOMEDRIVE%

**$envHomePath**  %HOMEPATH%

**$envHomeShare**  %HOMESHARE%

**$envLocalAppData** %LOCALAPPDATA%

**$envLogonServer** %LOGONSERVER%

**$envProgramFiles**  %PROGRAMFILES%

**$envProgramFilesX86** %PROGRAMFILES(X86)%

**$envProgramData** %PROGRAMDATA%

**$envPublic** %PUBLIC% (Windows Vista or later only)

**$envSystemDrive** %SYSTEMDRIVE%

**$envSystemRoot %**SYSTEMROOT%

**$envTemp** %TEMP%

**$envUserDNSDomain %**USERDNSDOMAIN%

**$envUserDomain** %USERDOMAIN%

**$envUserName** %USERNAME%

**$envUserProfile** %USERPROFILE%

**$envWinDir** %WINDIR%

# 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, e.g. HKLM is converted to "HKEY\_LOCAL\_MACHINE" and prepends "Registry::" to the path

**Parameter**  : Key

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

**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}"

### Copy-File

**Synopsis** : Function to copy a file to a destination path.

**Description**:

Function to copy a file to a destination path.

**Parameter**  : Path

Path of the file you want 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"

### 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.

Sets default switches to be passed to msiexec based on the preferences in the XML configuration file, e.g. "REBOOT=ReallySuppress /QB!"

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

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

MSI file.

**Parameter**  : Action

The action to perform ["Install","Uninstall","Patch","Repair","ActiveSetup"]

Path

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

Transform

The name of the transform file(s). The transform file is expected to be in the same directory as the MSI file.

Parameters

Overrides the default parameters specified in the XML configuration file. Install default is "REBOOT=ReallySuppress /QB!", uninstall default is

"REBOOT=ReallySuppress /QN"

LogName

Overrides the default log file name.

The default log file name is generated from the MSI file name or for uninstallations, the product code is resolved to the displayname and version of the

application.

WorkingDirectory

Overrides the working directory.

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

ContinueOnError

Continue if an exit code is returned by msiexec that is not recognised 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>Execute-MSI -Action Uninstall -Path "{26923b43-4d38-484f-9b9e-de460746276c}"

Uninstalls an MSI using a product code

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

C:\PS>Execute-MSI -Action Patch -Path "Adobe\_Reader\_11.0.3\_EN.msp"

Installs an MSP

### Execute-Process

**Synopsis** : Function to 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**  : FilePath

Path of the file you want to execute.

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 above.

Arguments

Arguments to be passed to the executable

WindowStyle

Style of the window of the process executed: "Normal","Hidden","Maximized","Minimized" [Default is "Normal"]

WorkingDirectory

The working directory used for executing the process.

Defaults to the directory of the file being executed.

NoWait

PassThru

Returns 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 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 is 600 seconds (10 minutes).

IgnoreExitCodes

List the exit codes you want to ignore.

ContinueOnError

Continue if an exit code is returned by the process that is not recognised by the App Deploy Toolkit.

**Examples** :

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

C:\PS>Execute-Process -FilePath "uninstall\_flash\_player\_64bit.exe" -Arguments "/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 -FilePath "$dirFiles\Bin\setup.exe" -Arguments "/S" -WindowStyle Hidden

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

C:\PS>Execute-Process -FilePath "setup.exe" -Arguments "/S" -IgnoreExitCodes "1,2"

### Exit-Script

**Synopsis** : This function exits the scripts, performs cleanup actions and passes an exit code to the parent process.

**Description**:

This function should always be used when exiting the script, to ensure cleanup actions are performed.

This function performs cleanup actions, such as closing down dialogs and unblocking blocked applications.

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

The function determines what exit code to pass to the parent process depending on the 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 "$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-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-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-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-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.

**Parameter**  :

Name

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

Exact

Specifies to only match the exact name of the application.

ProductCode

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

**Examples** :

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

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

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

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

### 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. 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)

ContinueOnError

Continue if an error is encountered

**Examples** :

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

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

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

C:\PS>Get-RegistryKey "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\iexplore.exe"

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

C:\PS>Get-RegistryKey "HKLM:\Software\Wow6432Node\Microsoft\Microsoft SQL Server Compact Edition\v3.5" -Value "Version"

### Get-ScheduledTask

**Synopsis** : Retrieves a list of the scheduled tasks on the local computer

**Description**:

Retrieves a list of the scheduled tasks on the local computer and returns them as an array

**Parameter**  : ContinueOnError

Continue if an error is encountered

**Examples** :

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

C:\PS>Get-ScheduledTask

### Install-MSUpdates

**Synopsis** : Installs all Microsft Updates in a given directory

**Description**:

Installs all Microsft Updates in a given directory of type ".exe", ".msu" or ".msp"

**Parameter**  : Directory

Directory containing the updates

**Examples** :

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

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

### Install-SCCMSoftwareUpdates

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

**Description**:

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

This function can take several minutes to run

**Parameter**  : ContinueOnError

Continue if an error is encountered

**Examples** :

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

C:\PS>Install-SCCMSoftwareUpdates

### Invoke-SCCMTask

**Synopsis** : Triggers SCCM to invoke the relevant task

**Description**:

Triggers SCCM to invoke the relevant task

**Parameter**  : ScheduleID

ScheduleId

ContinueOnError

Continue if an error is encountered

**Examples** :

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

C:\PS>Invoke-SCCMTask "SoftwareUpdatesScan"

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

C:\PS>Invoke-SCCMTask

### New-Folder

**Synopsis** : Function to create a new folder.

**Description**:

Function to create a new folder if it does not exist.

**Parameter**  : Path

Path of the folder you want to create.

ContinueOnError

Continue if an error is encountered

**Examples** :

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

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

### New-Shortcut

**Synopsis** : Creates a new shortcut .lnk or .url file, which can be used for example on the start menu.

**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

Description

**Description**of the shortcut

WorkingDirectory

Working Directory to be used for the target path

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** : Forces the Windows Exporer Shell to refresh, which causes desktop icons to be reloaded

**Description**:

Forces the Windows Exporer Shell to refresh, which causes desktop icons to be reloaded.

Informs the Explorer Shell to refresh its settings after you change registry values or other settings to avoid a reboot.

**Parameter**  : ContinueOnError

Continue if an error is encountered

**Examples** :

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

C:\PS>Refresh-Desktop

### Register-DLL

**Synopsis** : Registers a DLL file

**Description**:

Registers a DLL file using regsvr32.exe

**Parameter**  : FilePath

Path to the DLL file

ContinueOnError

Continue if an error is encountered

**Examples** :

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

C:\PS>Register-DLL "$envProgramFiles\Documentum\Shared\DcTLSFileToDMSComp.dll"

### Remove-File

**Synopsis** : Function to remove a file or all files recursively in a given path.

**Description**:

Function to remove a file or all files recursively in a given path.

**Parameter**  : Path

Path of the file you want to remove

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** : Function to remove a folder and files if they exist.

**Description**:

Function to remove a folder and all files recursively in a given path.

**Parameter**  : Path

Path of the folder you want to remove.

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 you want to uninstall. Performs a wildcard match on the application display name.

Exact

Specifies to only match the exact name of the application.

ContinueOnError

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

**Examples** :

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

C:\PS>Remove-MSIApplications "Adobe Flash"

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

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

C:\PS>Remove-MSIApplications "Adobe"

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

### 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

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"

### Send-Keys

**Synopsis** : Send a sequence of keys to an application window

**Description**:

Send a sequence of keys to an application window. Refer to MS documentation for more details on key sequences:

http://msdn.microsoft.com/en-us/library/System.Windows.Forms.SendKeys(v=vs.100).aspx

**Parameter**  : WindowTitle

The title of the application window. This can be a partial title.

**Parameter**  : Keys

The sequence of keys to send

**Parameter**  : WaitSeconds

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

**Examples** :

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

Send-Keys "foobar - Notepad" "Hello world" -WaitSeconds 5

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

**Examples** :

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

New-Item foobar.txt -ItemType File

notepad foobar.txt

Send-Keys "foobar - Notepad" "Hello world{ENTER}Ciao mondo{ENTER}" -WaitSeconds 1

Send-Keys "foobar - Notepad" "^s"

This command sequence creates a new text file called foobar.txt, opens the file using notepad.exe, writes some text, and saves the file using notepad.

### 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

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: "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 or value data or sets the same if it does not already exist.

**Description**:

Creates a registry key name, value or value data or sets the same if it does not already exist.

**Parameter**  : Key

The registry key path

Name

The value name

Value

The value data

Type

The type of registry value to create or set [Default is "String"

Acceptable values are: "Binary","DWord","ExpandString","MultiString","None","QWord","String","Unknown"

Object type: [Microsoft.Win32.RegistryValueKind]

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

### 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

Icon to be used [Default is Info]

Accepted values: 'Error', 'Info', 'None', 'Warning'

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** : This function displays a custom dialog box with optional title, buttons, icon and timeout.

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

**Description**:

This function displays 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 [Default is "OK"]

"OK"

"OKCancel"

"AbortRetryIgnore"

"YesNoCancel"

"YesNo"

"RetryCancel"

"CancelTryAgainContinue"

DefaultButton

The Default button that is selected [Default is "First"]

"First"

"Second"

"Third"

Icon

Icon to display on the dialog box [Default is "None"]

Acceptable valures are: "None", "Stop", "Question", "Exclamation", "Information",

Timeout

Timeout period in seconds before automatically closing the dialog box with the return message "Timeout" [Default the 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 is 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 mintues. 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 is just below top, centered]

TopMost

Specificies whether the progress window should be topmost [default is true]

**Examples** :

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

C:\PS>Show-InstallationProgress

Uses the default status message from the XML configuration file.

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

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

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

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

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

C:\PS>Show-InstallationProgress "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 is the application installation name]

Message

Message text to be included in the prompt

MessageAlignment

Alignment of the message text (Left,Center,Right) [Default is Center]

ButtonRightText

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

ButtonLeftText

Show a button on the left 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 ("Application","Asterisk","Error","Exclamation","Hand","Information","None","Question","Shield","Warning","WinLogo") [Default is "None"]

NoWait

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

Timeout

Specifies the period in seconds after which the prompt should timeout [Default is the UI timeout value set in the config XML file]

ExitOnTimeout

Specifies whether to exit the script if the UI times out. [Default is 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 customise 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** : This function provides 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:

Close the specified running applications, or optionally close the applications without showing a prompt (using the -Silent" switch).

Defer the installation a certain number of times, for a certain number of days or until a deadline is reached.

Countdown until applications are automatically closed.

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 function examples for more information.

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.

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.

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 is true]

**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 --------------------------

Show-InstallationWelcome -CloseApps "winword.exe,msaccess.exe,excel.exe" -PersistPrompt

Prompt the user to close Word, MSAccess and Excel if the processes match the exact name specified (use .exe for exact matches).

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.

### Test-Battery

**Synopsis** : Tests whether the local machine is running on battery

**Description**:

Tests whether the local machine is running on battery and returns true/false

**Parameter**  :

**Examples** :

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

C:\PS>Test-Battery

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

C:\PS>

### Test-MSUpdates

**Synopsis** : Test whether an Microsoft Windows update is installed

**Description**:

Test whether an Microsoft Windows update is installed

**Parameter**  : KBNumber

KBNumber

**Examples** :

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

C:\PS>Test-MSUpdates "KB2549864"

### Test-NetworkConnection

**Synopsis** : Tests for an active network connection

**Description**:

Tests for an active network connection by querying the Win32\_NetworkAdapter WMI class.

**Parameter**  :

**Examples** :

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

C:\PS>Test-NetworkConnection

### Test-PowerPoint

**Synopsis** : Tests whether Power point is running in presentation mode

**Description**:

Tests whether Power point is running in presentation mode

**Parameter**  :

**Examples** :

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

C:\PS>Test-PowerPoint

### Unregister-DLL

**Synopsis** : Unregisters a DLL file

**Description**:

Unregisters a DLL file using regsvr32.exe

**Parameter**  : FilePath

Path to the DLL file

ContinueOnError

Continue if an error is encountered

**Examples** :

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

C:\PS>Unregister-DLL "$envProgramFiles\Documentum\Shared\DcTLSFileToDMSComp.dll"

### 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** : Writes output to the console and log file simultaneously

**Description**:

This functions outputs text to the console and to the log file specified in the XML configuration.

The date, time and installation phase is pre-pended to the text, e.g. [30-07-2013 11:27:07] [Initialization] "Deploy Application script version is [2.0.0]"

**Parameter**  : Text

The text to display in the console and to write to the log file

PassThru

Passes the text back to the PowerShell pipeline

**Examples** :

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

C:\PS>Write-Log -Text "This is a custom message..."