**Notes on PSADT / SCCM**

PSADT is a god-send for anyone deploying applications via SCCM.

To install just run the script:

Deploy-Application.ps1

# if you want to be explicit (I like to be)

Deploy-Application.ps1 -DeploymentType 'Install'

# or even

Deploy-Application.ps1 'Install'

# silent variant

Deploy-Application.ps1 -DeployMode 'Silent'

To uninstall:

Deploy-Application.ps1 -DeploymentType 'Uninstall'

# or even

Deploy-Application.ps1 'Uninstall'

# silent variant

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

You can deploy in the following -DeployMode: Interactive (the default) or Silent or Noninteractive (very silent?).

When calling via SCCM you can use the Deploy-Application.exe helper instead. This will launch the script via PowerShell. Thus do the following for example:

Deploy-Application.exe -DeploymentType "Install" -DeployMode "Silent"

As opposed to:

powershell.exe -Command "& { & '.\Deploy-Application.ps1' -DeployMode 'Silent'; Exit $LastExitCode }"

If you decide to rename Deploy-Application.ps1 or have another copy in the same folder you could run it thus:

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

# Or a longer format

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

'Uninstall'

By default if there’s a reboot requirement from the MSI this is passed on to the calling process. You can suppress it:

|  |  |
| --- | --- |
|  | -AllowRebootPassThru $false |

Logs are stored at C:\Windows\Logs\Software.

The only file you need to modify with PSADT is the Deploy-Application.ps1 file. Apart from that there are Files and SupportFiles folders – use the former for storing any MSI and setup files, use the latter for any other files you want to copy over to the target machine. Easy peasy. All other files can be ignored, but you can tweak them if you want to customize PSADT further such as add a logo and change colours etc.

There’s a Zero Config MSI install method. This means you don’t modify the Deploy-Application.ps` file at all. You put the MSI in the Files folder. (Note: you can have only one MSI, so if more than one is put there only the first is used). You put the corresponding MST file (if any) in the Files folder. It must have the same name as the MSI, but with an MST extension of course. Finally you put any patches (MSP files) in the same Files folder – there can be more than one of these and they are installed in alphabetical order (so rename the files if order matters). All you do then is run the Deploy-Application.ps1 script.

The beauty of PSADT is that it gives you a ton of variables and helper functions you can use in the script.

You can determine if you are on a server for instance via the $IsServerOS variable, refer a user’s Desktop via $envUserDesktop or the profile itself via $envUserProfile, even things like the public desktop via $envCommonDesktop. These are just some examples, and nothing you can’t do in a longer way via PowerShell, but using PSADT gives you all these for free (which is always the point of a framework or language – give you things for free so you can focus on the bigger stuff). You can refer to the Files folder via $dirFiles and SupportFiles via $dirSupportFiles. For example the following uses a function provided by PSADT to copy over some files to C:\Windows:

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

An especially useful function is the Execute-MSI one. Here’s me using it to install Teams:

Execute-MSI -Action 'Install' -Path "$dirFiles\Teams\_windows\_x64.msi" -Parameters '/qn /NORESTART ALLUSER=1 ALLUSERS=1 OPTIONS="noAutoStart=true"'

Notice I can specify:

1. an action (-Action 'Install', -Action 'Uninstall', Action 'Repair', -Action 'Patch', -Action 'ActiveSetup'),
2. parameters to the msixec process (the -Parameters switch overrides the defaults, but we can do -AddParameters to add instead or -SecureParameters to not show the paramters in any logs),
3. transforms (not shown above but use the -Transform switch),
4. any logging options via -LoggingOptions (default is /l\*v to the default logging path).

Some useful functions are the Get-UserProfiles which lists all the userprofiles on the machine (and the Profilepath property can be used to get the path of the profile) or Invoke-HKCURegistrySettingsForAllUsers which can do a registry operation against all the HKCU registry keys. Here’s me using the latter to delete certain Teams related keys:

# a block of registry cmdlets (provided by PSADT) that I want to run

# note these target HKCU. And I have to add the -SID part for user later on

[scriptblock]$HKCURegistryChanges = {

Remove-RegistryKey -Key 'HKCU\Software\Microsoft\Office\Teams' -SID $UserProfile.SID

Remove-RegistryKey -Key 'HKCU\Software\Microsoft\Windows\CurrentVersion\Uninstall\Teams' -SID $UserProfile.SID

}

# I run the above script block against the HKCU for all users

Invoke-HKCURegistrySettingsForAllUsers -RegistrySettings $HKCURegistryChanges