



ceterion Packaging Framework

Admin Guide

Overview	2
New Features	3
Optionally available	3
System Requirements.....	4
Licensing	4
New Functions, added by ceterion.....	4
Existing or changed functions.....	5
Removed functions:	6
Getting started guide	7
How to create a package	12
JSON File.....	14

Overview

The ceterion Packaging Framework has been developed to package Applications-Setups and deploying with several Deployment Systems. The cmPF is based on the "PowerShell App Deployment Toolkit 3.7.0" <http://psappdeploytoolkit.com/>.

It contains parts of the original toolkit and provides a number of extensions and changes. The modification includes for example a conversion from a simple included script into a PowerShell module and is extended with some additional functions and variables we missed in the original implementation.

New Features

- Framework has been changed into a "Module" and therefore now provides all the benefits, that go with this. For Example
 - IntelliSense for commands and variables Context-sensitive
 - Online help (using F1 button)
 - Using "commands register" in PowerShell ISE and integrated command help
- Basic simplification, streamlining and updating
- Concept for abstraction of metadata and parameterization
- Simple parameter handling based on JSON-files
- Including encryption for passwords, license keys etc.
- Flexible, configurable naming scheme
- Package validation based on a configurable set of rules
- Advanced error handling and logging
- Renaming variables and functions
- Expansion of variables (already available during packaging)
- New commands and extensions in known commands
- Install multiple packages including reboots etc. without any deployment system for such as testing
- Support Citrix environments with automatic publishing, NTFS-Security and Start Menu population
- Support Deployment Systems like SCCM, ASG Cloud Shaper and many more
- Expandable with your own modules
- Small and clean installation scripts
- Package Folders and Installation scripts can be generated by command

Optionally available

- SCCM/MECM Package Importer
- VMware Workspace ONE UEM Package Importer
- Create Task sequences based on templates
- Hundreds of templates for applications and operating system components (Adobe Reader, MS Office etc.)
- Packaging-Training for beginners and advanced users
- Citrix installation scripts with automated publishing

System Requirements

Requirements on target systems:

- Windows PowerShell 5.1
- local admin rights on target system
- PowerShell execution policy (packaging)

System Requirements:

- like target system
- Editor with PowerShell support (e.g. Microsoft PowerShell ISE and/or Microsoft VS-Code)

Licensing

The ceterion modular Packaging Framework is based on PowerShell App Deployment Toolkit and provided under the Microsoft Public License:

<https://msdn.microsoft.com/en-us/library/ff648068.aspx>

New Functions, added by ceterion

- Add-Font
- Add-Path
- Convert-Base64
- ConvertFrom-AAPINI
- ConvertFrom-Ini
- ConvertFrom-IniFiletoObjectCollection
- ConvertTo-Ini
- Expand-Variable
- Get-EnvironmentVariable
- Get-FileVerb
- Get-Parameter
- Get-Path
- Import-RegFile
- Initialize-Script
- Install-DeployPackageService
- Install-MultiplePackages
- Invoke-Encryption
- Invoke-FileVerb

- Invoke-InstallOrRemoveAssembly
- New-File
- New-LayoutmodificationXML
- New-Package
- Remove-EnvironmentVariable
- Remove-Font
- Remove-Path
- Set-AutoAdminLogon
- Set-DisableLogging
- Set-EnvironmentVariable
- Set-InstallPhase
- Start-NSISWrapper
- Test-IsGroupMember
- Test-Package
- Test-PackageName
- Update-FilePermission
- Update-FolderPermission
- Update-FrameworkInPackages
- Update-Ownership
- Update-PrinterPermission
- Update-RegistryPermission
- Update-SessionEnvironmentVariables

Existing or changed functions

- Close-InstallationProgress
- Copy-File
- Disable-TerminalServerInstallMode
- Enable-TerminalServerInstallMode
- Exit-Script
- Get-FileVersion
- Get-FreeDiskSpace
- Get-HardwarePlatform
- Get-IniValue
- Get-InstalledApplication
- Get-LoggedOnUser
- Get-PendingReboot
- Get-RegistryKey
- Get-ServiceStartMode
- Install-MSUpdates
- Install-SCCMSoftwareUpdates

- Invoke-RegisterOrUnregisterDLL
- Invoke-SCCMTask
- New-Folder
- New-MsiTransform
- New-Shortcut
- Remove-File
- Remove-Folder
- Remove-MSIApplications
- Remove-RegistryKey
- Resolve-Error
- Set-ActiveSetup
- Set-IniValue
- Set-PinnedApplication
- Set-RegistryKey
- Set-ServiceStartMode
- Show-BalloonTip
- Show-DialogBox
- Show-InstallationProgress
- Show-InstallationPrompt
- Show-InstallationRestartPrompt
- Show-InstallationWelcome
- Start-MSI
- Start-Program
- Start-ServiceAndDependencies
- Stop-ServiceAndDependencies
- Test-MSUpdates
- Test-Ping
- Test-RegistryKey
- Test-ServiceExists
- Update-Desktop
- Write-FunctionHeaderOrFooter
- Write-Log

Removed functions:

- Execute-ProcessAsUser
- Invoke-HKCUREgistrySettingsForAllUsers
- Test-Battery
- Test-NetworkConnection
- Test-PowerPoint

Getting started guide

First make sure, you start your PowerShell session with local admin permissions. When UAC is enabled, make sure to start your PowerShell session in elevated mode.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Windows\system32> 
```

Also make sure your PowerShell execution policy is configured to run scripts, i.e. you can configure it with this PowerShell command:

1. Set-ExecutionPolicy RemoteSigned (depending on your scenario)

```
PS C:\Windows\system32> Set-ExecutionPolicy RemoteSigned

Ausführungsrichtlinie ändern
Die Ausführungsrichtlinie trägt zum Schutz vor nicht vertrauenswürdigen Skripts bei. Wenn Sie die Ausführungsrichtlinie
ändern, sind Sie möglicherweise den im Hilfethema "about_Execution_Policies" unter
"https://go.microsoft.com/fwlink/?LinkID=135170" beschriebenen Sicherheitsrisiken ausgesetzt. Möchten Sie die
Ausführungsrichtlinie ändern?
[J] Ja [A] Ja, alle [N] Nein [K] Nein, keine [H] Anhalten [?] Hilfe (Standard ist "N"): A
```

To import the module, use the following PowerShell command:

2. Import-Module PackagingFramework

```
Administrator: Windows PowerShell
PS C:\Windows\system32> Import-Module PackagingFramework
PS C:\Windows\system32> 
```



To Initialize the runtime variables, use the following PowerShell command:

3. Initialize-Script

```
Administrator: Windows PowerShell
PS C:\Windows\system32> Initialize-Script
[10-20-2017 14:05:13.576] [Initialization] [PackagingFramework] :: *****
*****
[10-20-2017 14:05:13.654] [Initialization] [PackagingFramework] :: Import Extension [C:\Program Files\WindowsPowerShell\
Modules\PackagingFramework\..\PackagingFrameworkExtension\PackagingFrameworkExtension.psd1]
[10-20-2017 14:05:13.669] [Initialization] [PackagingFramework] :: PackagingFramework module version is [1.0.0.0]
[10-20-2017 14:05:13.685] [Initialization] [PackagingFramework] :: PackagingFramework module base [C:\Program Files\Wind
owsPowerShell\Modules\PackagingFramework]
[10-20-2017 14:05:13.685] [Initialization] [PackagingFramework] :: PackagingFrameworkExtension module version is [1.0.0.
0]
[10-20-2017 14:05:13.685] [Initialization] [PackagingFramework] :: PackagingFrameworkExtension module base [C:\Program F
iles\WindowsPowerShell\Modules\PackagingFrameworkExtension]
[10-20-2017 14:05:13.685] [Initialization] [PackagingFramework] :: Computer Name is [DESKTOP-H4187DA]
[10-20-2017 14:05:13.685] [Initialization] [PackagingFramework] :: Current User is [DESKTOP-H4187DA\ceterion]
[10-20-2017 14:05:13.685] [Initialization] [PackagingFramework] :: OS Version is [Microsoft Windows 10 Enterprise Evalua
tion 64-bit 10.0.16299]
[10-20-2017 14:05:13.701] [Initialization] [PackagingFramework] :: OS Type is [Workstation]
[10-20-2017 14:05:13.763] [Initialization] [PackagingFramework] :: Current Culture is [de-DE] and UI language is [DE]
[10-20-2017 14:05:13.841] [Initialization] [PackagingFramework] :: Hardware Platform is [Virtual:Hyper-V]
[10-20-2017 14:05:13.857] [Initialization] [PackagingFramework] :: PowerShell Host is [ConsoleHost] with version [5.1.16
299.15]
[10-20-2017 14:05:13.857] [Initialization] [PackagingFramework] :: PowerShell Version is [5.1.16299.15 x64]
[10-20-2017 14:05:13.857] [Initialization] [PackagingFramework] :: PowerShell CLR (.NET) version is [4.0.30319.42000]
[10-20-2017 14:05:13.857] [Initialization] [PackagingFramework] :: Installation is running in [Install] type.
[10-20-2017 14:05:13.857] [Initialization] [PackagingFramework] :: Initialize-Script completed.
[10-20-2017 14:05:13.873] [Initialization] [PackagingFramework] :: *****
*****
PS C:\Windows\system32>
```


The get a list of all included commands use the following PowerShell command:

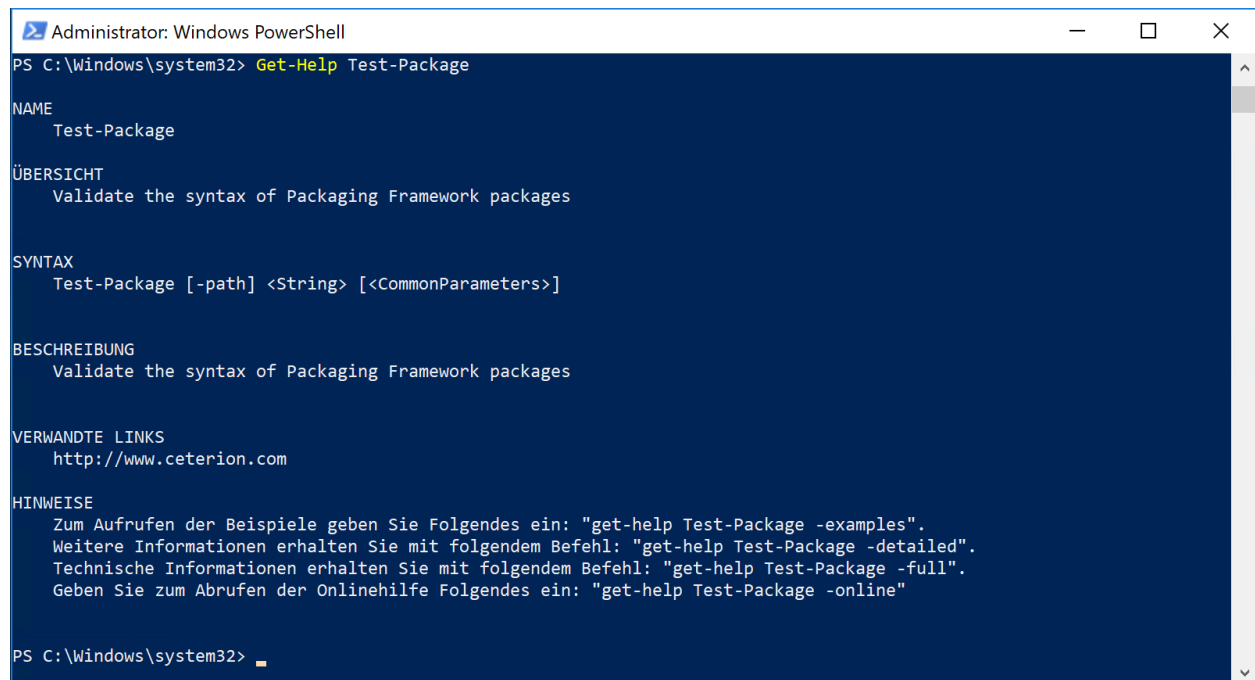
Get-Command -Module PackagingFramework

```
Administrator: Windows PowerShell
PS C:\Windows\system32> Get-Command -Module PackagingFramework
```

CommandType	Name	Version	Source
Alias	Register-DLL	1.0.0.0	PackagingFramework
Alias	Unregister-DLL	1.0.0.0	PackagingFramework
Function	Add-Font	1.0.0.0	PackagingFramework
Function	Add-Path	1.0.0.0	PackagingFramework
Function	Convert-Base64	1.0.0.0	PackagingFramework
Function	ConvertFrom-AAPINI	1.0.0.0	PackagingFramework
Function	ConvertFrom-Ini	1.0.0.0	PackagingFramework
Function	ConvertFrom-IniFiletoObjectCollection	1.0.0.0	PackagingFramework
Function	ConvertTo-Ini	1.0.0.0	PackagingFramework
Function	ConvertTo-NTAccountOrSID	1.0.0.0	PackagingFramework
Function	Copy-File	1.0.0.0	PackagingFramework
Function	Disable-TerminalServerInstallMode	1.0.0.0	PackagingFramework
Function	Edit-StringInFile	1.0.0.0	PackagingFramework
Function	Enable-TerminalServerInstallMode	1.0.0.0	PackagingFramework
Function	Exit-Script	1.0.0.0	PackagingFramework
Function	Expand-Variable	1.0.0.0	PackagingFramework
Function	Get-EnvironmentVariable	1.0.0.0	PackagingFramework
Function	Get-FileVerb	1.0.0.0	PackagingFramework
Function	Get-FileVersion	1.0.0.0	PackagingFramework
Function	Get-FreeDiskSpace	1.0.0.0	PackagingFramework
Function	Get-HardwarePlatform	1.0.0.0	PackagingFramework
Function	Get-IniValue	1.0.0.0	PackagingFramework
Function	Get-InstalledApplication	1.0.0.0	PackagingFramework
Function	Get-LoggedOnUser	1.0.0.0	PackagingFramework

To get help for the individual PowerShell commands of the module use the following PowerShell command:

Get-Help <Command>



```
Administrator: Windows PowerShell
PS C:\Windows\system32> Get-Help Test-Package

NAME
    Test-Package

ÜBERSICHT
    Validate the syntax of Packaging Framework packages

SYNTAX
    Test-Package [-path] <String> [<CommonParameters>]

BESCHREIBUNG
    Validate the syntax of Packaging Framework packages

VERWANDTE LINKS
    http://www.ceterion.com

HINWEISE
    Zum Aufrufen der Beispiele geben Sie Folgendes ein: "get-help Test-Package -examples".
    Weitere Informationen erhalten Sie mit folgendem Befehl: "get-help Test-Package -detailed".
    Technische Informationen erhalten Sie mit folgendem Befehl: "get-help Test-Package -full".
    Geben Sie zum Abrufen der Onlinehilfe Folgendes ein: "get-help Test-Package -online"

PS C:\Windows\system32>
```

To get a full help of all included command use the following PowerShell command:

[Get-Command -Module PackagingFramework | Get-Help](#)

To get a list of all runtime variables use the following PowerShell command:

[Get-Variable | Out-GridView](#)

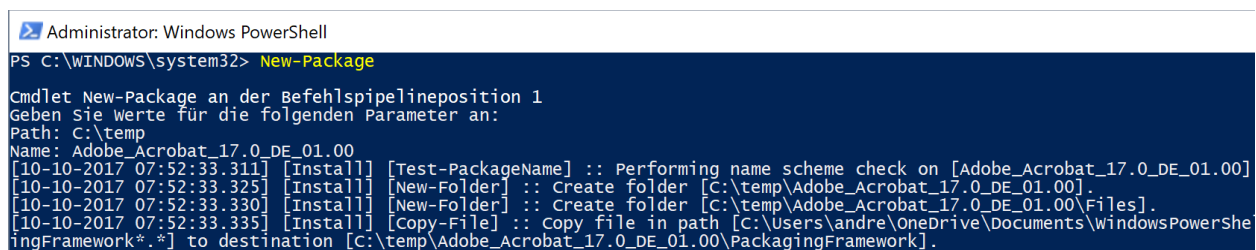
[Get-Variable | Out-GridView](#)

Filter	
+ Kriterien hinzufügen ▼	
Name	Value
CurrentProcessSID	S-1-5-21-1520544230-3956485659-3817253734-1001
CurrentProcessToken	System.Security.Principal.WindowsIdentity
CurrentTime	14:05:10
CurrentTimeZoneBias	02:00:00
CustomTypesFile	C:\Program Files\WindowsPowerShell\Modules\PackagingFrame...
DebugPreference	SilentlyContinue
DefaultUserProfile	C:\Users\Default
Error	{}
ErrorActionPreference	Continue
ErrorView	NormalView
ExampleVarFromExtension	Hello World
ExecutionContext	System.Management.Automation.EngineIntrinsics
false	False
Files	System.Management.Automation.InvocationInfo.MyCommand.D...
FormatEnumerationLimit	4
HOME	C:\Users\ceterion
HomeDrive	C:
HomePath	\Users\ceterion
HomeShare	
Host	System.Management.Automation.Internal.Host.InternalHost

How to create a package

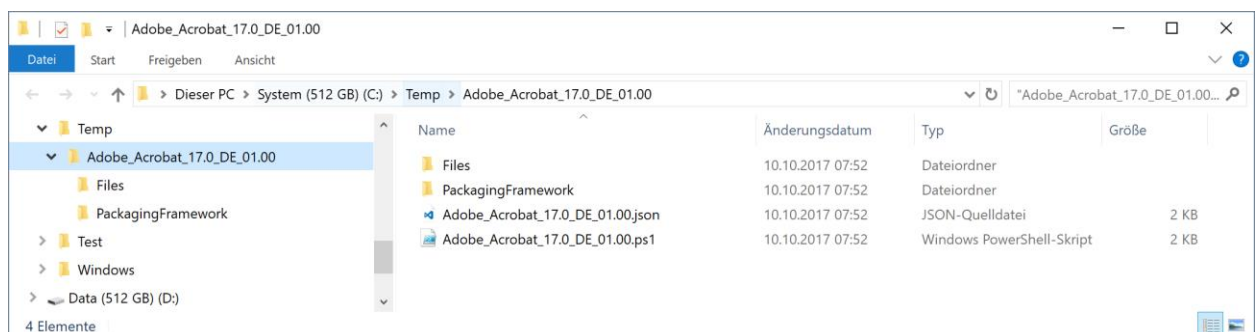
To create your first own package with cmPF, use the following PowerShell command:

`New-Package -Path C:\Temp -Name 'Adobe_Acrobat_17.0_DE_01.00'`
 (Specify the desired package path and the package name
 According to the name schema (customizable, in the delivery state
 from AppVendor, AppName, AppVersion, AppLanguage and package Version)



```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> New-Package

Cmdlet New-Package an der Befehlspipelineposition 1
Geben Sie Werte für die folgenden Parameter an:
Path: C:\temp
Name: Adobe_Acrobat_17.0_DE_01.00
[10-10-2017 07:52:33.311] [Install] [Test-PackageName] :: Performing name scheme check on [Adobe_Acrobat_17.0_DE_01.00]
[10-10-2017 07:52:33.325] [Install] [New-Folder] :: Create folder [C:\temp\Adobe_Acrobat_17.0_DE_01.00].
[10-10-2017 07:52:33.330] [Install] [New-Folder] :: Create folder [C:\temp\Adobe_Acrobat_17.0_DE_01.00\Files].
[10-10-2017 07:52:33.335] [Install] [Copy-File] :: copy file in path [C:\Users\andre\OneDrive\Documents\WindowsPowerShe
ingFramework*.ps1] to destination [C:\temp\Adobe_Acrobat_17.0_DE_01.00\PackagingFramework].
```



Open your `Adobe_Acrobat_17.0_DE_01.00.ps1`-file and edit:

```
[CmdletBinding()] Param ([Parameter(Mandatory=$false)] [ValidateSet('Install','Uninstall')]
[string]$DeploymentType='Install', [Parameter(Mandatory=$false)]
[ValidateSet('Interactive','Silent','NonInteractive')] [string]$DeployMode='Interactive')
Try {
    # Import Packaging Framework Module
    Import-Module PackagingFramework ; Initialize-Script
    # Install
    If ($deploymentType -ieq 'Install') {
        # <PLACE YOUR CODE HERE>
    }
    # Uninstall
    If ($deploymentType -ieq 'Uninstall') {
        # <PLACE YOUR CODE HERE>
    }
    # Call the exit-Script
    Exit-Script -ExitCode $mainExitCode
}
Catch { [int32]$mainExitCode = 60001; [string]$mainErrorMessage = "$(Resolve-Error)"; Write-Log -
Message $mainErrorMessage -Severity 3 -Source $PackagingFrameworkName ; Show-DialogBox -Text
$mainErrorMessage -Icon 'Stop' ; Exit-Script -ExitCode $mainExitCode }
```

```
[CmdletBinding()] Param ([Parameter(Mandatory=$false)] [ValidateSet('Install','Uninstall')]
[string]$DeploymentType='Install', [Parameter(Mandatory=$false)]
[ValidateSet('Interactive','Silent','NonInteractive')] [string]$DeployMode='Interactive')
Try {
    # Import Packaging Framework Module
    Import-Module PackagingFramework ; Initialize-Script
    # Install
    If ($deploymentType -ieq 'Install') {
        Start-MSI -Action 'Install' -Path "$Files\AcroRead.msi" -Parameters "AUTOUPDATE=NO"
    }
    # Uninstall
    If ($deploymentType -ieq 'Uninstall') {
        Start-MSI -Action 'Uninstall' -Path "$Files\AcroRead.msi"
    }
    # Call the exit-Script
    Exit-Script -ExitCode $mainExitCode
}
Catch { [int32]$mainExitCode = 60001; [string]$mainErrorMessage = "$(Resolve-Error)" ; Write-Log -
Message $mainErrorMessage -Severity 3 -Source $PackagingFrameworkName ; Show-DialogBox -Text
$mainErrorMessage -Icon 'Stop' ; Exit-Script -ExitCode $mainExitCode }
```

Copy the Adobe-Installation Files into the folder:
 "C:\Temp\Adobe_Acrobat_17.0_DE_01.00\Files"

Edit the Adobe_Acrobat_17.0_DE_01.00.json with Notepad or Visual Studio Code like this:

```
{
  "Package": {
    "PackageDate": „10.10.2017“,
    "PackageAuthor": „Your Name“,
    "PackageDescription": "Adobe Acrobat Reader 17.0"
  },
  "Applications": [
    {
      "AppName": "Acrobat Reader",
      "AppFolder": "Utilities",
      "AppCommandLineExecutable": "%ProgramFiles%\AcrobatReader\AcrobatReader.exe",
      "AppCommandLineArguments": "",
      "AppWorkingDirectory": "",
      "AppAccounts": []
    }
  ],
  "DetectionMethods": [
  ],
  "Dependencies": [],
  "Parameters": {},
  "Notes": [],
  "ChangeLog": [
    "Version 1.0 initial release"
  ]
}
```

JSON File

- Each package has its own individual JSON file which gets the same name as the package folder and script
- The JSON file is being read and processed during package execution.
- Start menu links or Citrix published apps are automatically created or prepared using publishing settings from json file
- information such as license keys, Groups, host names, as parameter related and uses
- The JSON file can also be centralized on a network share Centrally placed configuration files override default configurations within packages. This makes it possible, to use the same package for different customers/clients with different configurations

Sections in JSON Files:

- **Package** includes general metadata about the package such as description, author, date
- **Applications** is always to be filled when Start menu links (on client OS) or published Applications (Server OS/Citrix) have to be created. The creation of the Citrix Publ. Apps takes place further through the CitrixPublishing package (not included by default)
- **DetectionMethods** is for SCCM with the path to a uniquely identifiable file of the package to populate "dependencies". It's optional and contains information about prerequisites in form of a free text
- **Changelog** contains information about what changes have been made to the package in which version.
- **Parameters** is an optional section if the package is to be controlled flexibly by parameters at runtime
- **Notes** is optional and contains information similar to a readme.txt file in form of free text
- **PackageDate** Date of package creation or last modification
- **PackageAuthor** Name of the package creator
- **PackageDescription** Description,
- **PackageInstallName** Display name is used for example in dialogs (optional)
- **PackageInstallTitle** Display title is used for example in dialogs (optional)
- **AppName** Application name, e.g. Google Chrome
- **AppCommandLineExecutable** Path to the program file, e.g. %ProgramFiles%\Google\Chrome\chrome.exe

- **AppWorkingDirectory** working directory.
e.g. %ProgramFiles%\Google\Chrome
 - **AppFolder** Application Folder for Start menu or Citrix console, e.g. "Google"
 - **AppCommandLineArguments** e.g. www.ceterion.de
 - **AppIconSource** Icon file (if it does not come from the .exe file,
e.g. %ProgramFiles%\Google\Chrome\Chrome.ico)
 - **AppIconIndex** Indexnumber of the icon if there are multiple icons in a file, e.g. 5
 - **AppAccounts** User group, also several possible, e.g. Domain\Domain users
-
- A single JSON file can contain several applications in the applications section (see e.g. office package like winword.exe, excel.exe, powerpoint.exe)
 - There may also be multiple accounts in the accounts section of the applications to authorize multiple groups
 - You can also use other Citrix Publ. App properties are specified (see Citrix New-BrokerApplication PowerShell command, optional available)
 - The json-file can be used optionally to make packages more flexible, e.g. for:
 - Component selection (keyword ADDLOCAL)
 - flexible target directory (keyword INSTDIR)
 - different license keys for different users/customers
 - different Backend systems for different environments (e.g. Dev, pilot, prod)
 - switching on and off of Div. installation options or optimization parameters
 - can be encrypted if required (e.g. passwords)
 - within the package convenient access to the parameters via get parameter
 - Several clients, one package
 - you are able to use one and the same package multiple times for different customers/clients, by combining the installation logic (.PS1 file) with different JSON files containing specific configuration data for each installation
 - by default, the local \$PackageName.JSON that is contained in the package is used at runtime of the package.



- Optionally, this \$PackageName.JSON can also be used from different network shares to apply distinct configurations to any user/customer/client
- On the network share, there must be a folder which is named as the package itself. It must contain the JSON file with the file name \$PackageName.JSON
- This folder can optionally be used to store other customer-specific things such as license files, other configuration files or. MST files
- The path to the network share is defined by the registry key HKLM\Software\PackagingFramework\ConfigRepositoryShare
- Optionally, a user account and password (encrypted) can also be specified for access to the network share the package
- Ceterion_Packagingframeworkconfig_ 1.0 _ml_ 1.00 contains the appropriate logic to set it up on the target system

To customize the packaging framework to your needs please have a look at the module configuration file at:

'%ProgramFiles%\WindowsPowerShell\Modules\PackagingFramework\PackagingFramework.json'

When the "Example Package" option was selected while installing the setup, you will find the examples at:

'%MyDocuments%\Packaging Framework Examples'

It also contains a template (Ceterion_examplepackage_ 1.0 _en_ 01.00) with instructions for the contained commands and functions.


```
Ceterion_ExamplePackage_1.0_EN_01.00.ps1 X
215 | #region Execute...
243 |
244 | #region MSI...
280 |
281 | #region Files...
305 |
306 | #region Registry
307 |
308 |     # Retrieve a registry value
309 |     $Result = Get-RegistryKey -Key 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\I
310 |     if ($Result) {Write-Log "Result= $Result"}
311 |
312 |     # Retrieve multiple values from a registry key
313 |     $Result = Get-RegistryKey -Key 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\I
314 |     if ($Result) {
```

You can contact us at the following email address:

PackagingFramework@ceterion.com