Windows Client Management AG
Software Deployment Solutions

Scripting Framework PowerShell Toolkit

Manual – Version 6.0

1 About Scripting Framework 7

1.1 How to use Scripting Framework? 7
1.2 Facts 7
1.3 Environment integration 8
1.4 Scripting Framework Suite 8
1.5 Technical requirements 8
1.6 Optional requirements 8

2 Available Functions 9

2.1 SF-AppvInstall 9
2.2 SF-AppvUninstall 10
2.3 SF-AppxInstall 11
2.4 SF-AppxUninstall 12
2.5 SF-CD 13
2.6 SF-Copy 14
2.7 SF-Crypt 15
2.8 SF-Decrypt 16
2.9 SF-Delete 17
2.10 SF-DriverInstall 18
2.11 SF-DriverUninstall 19
2.12 SF-Exit 20
2.13 SF-ExtractArchive 21
2.14 SF-File 22
2.15 SF-FileSearch 23
2.16 SF-FileVersion 24
2.17 SF-FontInstall 25
2.18 SF-FontUninstall 26
2.19 SF-GroupMembership 27
2.20 SF-INIRead 28
2.21 SF-INIWrite 29
2.22 SF-Installed 30

2.23 SF-Language 31
2.24 SF-LoadVariables 33
2.25 SF-Lockscreen 34
2.26 SF-Log 36
2.27 SF-Logoff 37
2.28 SF-MD 38
2.29 SF-MessageBox 39
2.30 SF-MSIGetProperty 40
2.31 SF-MSIInstall 41
2.32 SF-MSIPatchInstall 42
2.33 SF-MSIRepair 43
2.34 SF-MSIUninstall 44
2.35 SF-MSIUninstallByDisplayName 45
2.36 SF-MSIXInstall 46
2.37 SF-MSIXUninstall 47
2.38 SF-NTFSPerm 48
2.39 SF-NuGetInstall 49
2.40 SF-NuGetUninstall 50
2.41 SF-Path 51
2.42 SF-PinnedApplication 52
2.43 SF-ProgramsAndFeatures 53
2.44 SF-RD 54
2.45 SF-ReloadVariables 55
2.46 SF-RemoveVariables 56
2.47 SF-Register32 57
2.48 SF-Register64 58
2.49 SF-RegisterFile32 59
2.50 SF-RegisterFile64 60
2.51 SF-RegPerm32 61
2.52 SF-RegPerm64 62
2.53 SF-RegRead32 63
2.54 SF-RegRead64 64

2.55 SF-RegSearch32 65
2.56 SF-RegSearch64 66
2.57 SF-Rename 68
2.58 SF-Replace 69
2.59 SF-Requirements 70
2.60 SF-Run 71
2.61 SF-Service 73
2.62 SF-ServiceInstall 74
2.63 SF-ServiceStatus 75
2.64 SF-Set 77
2.65 SF-Shortcut 78
2.66 SF-ShowBalloonTip 79
2.67 SF-ShowCustomNotification 80
2.68 SF-ShowInstallNotification 82
2.69 SF-ShowProgressBar 86
2.70 SF-ShowSCCM 87
2.71 SF-Supersedence 88
2.72 SF-SymbolicLink 90
2.73 SF-SystemReboot 91
2.74 SF-Taskkill 92
2.75 SF-TestMSUpdate 93
2.76 SF-Textfile 94
2.77 SF-UnRegisterFile32 96
2.78 SF-UnRegisterFile64 97
2.79 SF-Variables 98
2.80 SF-Wait 99
2.81 SF-WUSAInstall 100

3 Available Suite Tools 101

3.1 EXE Generator 102
3.1.1 Example Package To EXE 102
3.1.2 Example Script To EXE (Scripting Mode) 103
3.2 Help Tools 105

3.3 Manual English 105
3.4 Manual German 105
3.5 Intune Generator 106
3.6 REG File Converter 107
3.7 Remote Manager 108
3.7.1 Tool Bar 108
3.7.2 Tab Pages 109
3.8 SCCM Application Creator 113
3.9 Script Generator 114
3.9.1 Script window 115
3.9.2 The functional area 116
3.9.3 The menu bar 118
3.10 Support Collector Tool 120
3.11 Ultimate Package 121

4 Help and examples 123
4.1 Functions withing the script 123

5 Scripting Framework software package (Definition) 125
5.1 Package name (main folder) 125
5.2 Folder and file structure 125
5.2.1 User structure - In the folder „User“ 126
5.3 Details of the individual files 126
5.3.1 Install.exe 126
5.3.2 Install.ps1 126
5.3.3 Package.xml 128
5.3.4 Uninstall.exe 130
5.3.5 Uninstall.ps1 130
5.3.6 InstallUser.ps1 130

6 Scripting Mode 131
6.1 Application - Launcher 131
6.2 Application - App-V 132

7 Structure of the toolkit and further information 133
7.1 Files 133

7.2 Registry 135
7.2.1 Config 135
7.2.2 Inventory 137
7.2.3 Reboot 138
7.2.4 Variables 139
7.2.4.1 Machine - Default variables 139
7.2.4.2 Machine - Runtime variables 141
7.2.4.3 User - Default variables 141
7.2.4.4 User - Default variables (Active Directory) 142
7.2.4.5 User - Runtime variables 144
7.3 Command Line Options (Install.exe and Uninstall.exe) 145
7.4 Logs and Errorcodes 145
7.4.1 Log files 145
7.4.2 Return values 147
7.5 Loading of package variables for dynamic packages 148
7.6 Handling of user settings 149
7.6.1 Technical description 149
7.6.2 Procedure 150
7.6.3 Published application (Citrix) 150

8 Deployment and configuration 151
8.1 Set up the central share (optional) 151
8.2 Create a service account in AD (optional) 151
8.3 Defining the future package name (identifier) 151
8.4 Perform initial configuration 152
8.5 Installation 155
8.5.1 Manual installation 155
8.5.2 Using software distribution (General) 156
8.5.3 Using software distribution (Microsoft SCCM) 156
8.5.3.1 Task Sequence Integration 168
8.6 Automatic update of Scripting Framework (AutoUpdate) 169
8.6.1 Operation 169
8.6.2 Required share structure 170
8.7 Installation engineer extension (for the development of packages) 171

9 Find out more and more areas of application 172
9.1 Immediate user installation 172
9.2 Notification Center (user notifications) 172
9.3 Installing drivers 173

9.4 Load licensed fonts only for the user session (Citrix) 174

1 About Scripting Framework

1.1 How to use Scripting Framework?

The Windows PowerShell tool kit offers you numerous functions for software packaging, general scripting and final deployment. It simplifies your company in the complex challenges which you always encounter more in this area. With this toolkit you can replace your VBScripts, WiseScripts, Batches, etc. with a multifacted and expandable tool, that allows you to increase the quality of your product.

Due to the large number of existing and specially tailored functions, as well as the structure, you can achieve a uniform packaging and standardization. The entire logic (eg, language switches, settings like server name, etc.) can be dynamically and centrally stored in the Scripting Framework package. The available functions are constructed simple and understandable and can be used successfully without PowerShell knowledge. This flexibility allows you to make uniform packages for various mandates or locations with different settings. Hybrid packages which on server (Citrix) and clients are used with different settings and architectures (x86 / x64), are easy to realize. Thus you create only a single package in the future, which covers the entire requirements.
Another unique feature is the simplicity of how to implement existing user settings within seconds in a package.

All existing functions are also outside of a Scripting Framework package available - in the so-called "Scripting mode". This allows the toolkit to be used very flexibly in various scenarios, such as dynamic App-V packages which needs different settings depending on the location.

1.2 Facts

- **Packager -** You want a setting within an application? No problem, our Packager helps you to find the appropriate registry keys or files.
- **Easy to use -** you don't need PowerShell knowledge. A large number of functions, especially in the field of packaging and software deployment.
- **Quality -** create packages and scripts with a uniform appearance, regardless of complexity.
- **Future -** MS PowerShell is established and the future in scripting - use it also for the software packaging!
- **Performance -** Over 60 functions for installation and removal of applications, deleting files and folders included the automatic closing of the active executables and open trade, writing of registry values (32-bit and 64-bit), copy of files, installation and uninstallation of AppV packages, keyboard shortcuts, fonts, services, etc. it's of course also possible to add your own PowerShell scripting code within the framework packages to use.
- **User Settings -** With the optional user script, you manage complex settings easily. Quick and easy you can write registry values, copy files to the user profile, create shortcuts, write INI file entries, etc. In addition, the user settings applied after the package installation automatically by Scripting Framework (without a user log off / log on) and this for all registered users on the device.
- **Dynamic -** The creation of a single package, for different locations, for example, use with different server names.
- **Usability -** The simultaneous use on Citrix and clients, regardless of whether it is a 32-bit or 64-bit operating system.
- **Independence -** All software packages that are created with Scripting Framework can be provided easily and use with a software distribution of your choice (manufacturer-independent).
- **Integratable -** Scripting framework is easily integrated into your existing environment

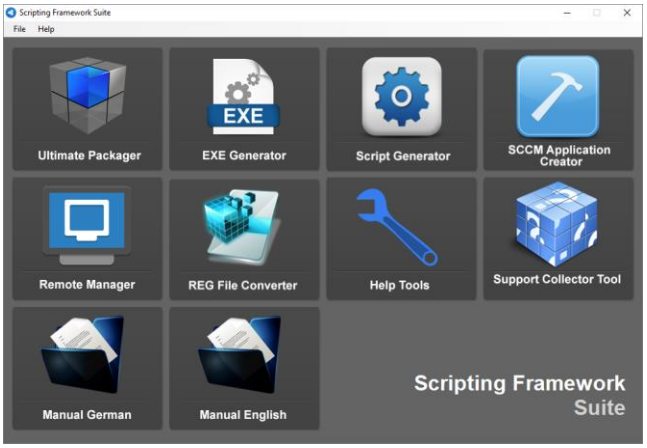
- **Use proven -** This toolkit is successfully used by our customers.

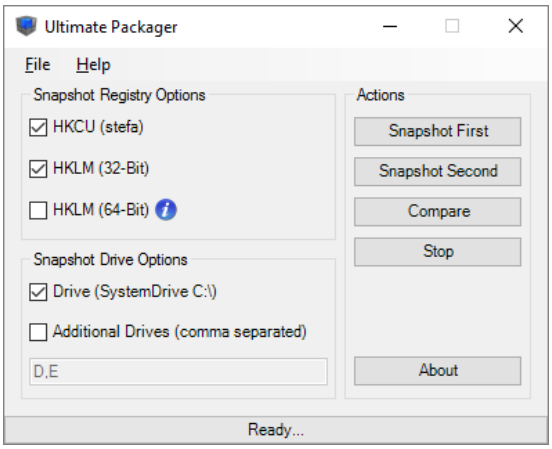
1.3 Environment integration

- Scripting Framework is with all of us known software deployment solutions compatible. A Porting of existing Scripting Framework packages on a new distribution system is easily possible and thus offers you maximum flexibility.
- Optimized for use with the Microsoft System Center Configuration Manager (SCCM).
- There are standard return codes (analog Windows Installer) which returned and can be analyzed.
- Can be integrated into the App-V packages using scripting Framework "Scripting Mode" (Scripting and Embedded Scripting for AppV 5.0)

1.4 Scripting Framework Suite

The Scripting Framework Suite includes all tools which are available as well as the Script Engine and which support you optimally. The suite includes for example the Ultimate Packager and the Script Generator, which allows you to quickly and easily create software packages based on the Scripting Framework functions.





1.5 Technical requirements

- Microsoft .Net Framework 4.8
- Microsoft PowerShell 5.x

1.6 Optional requirements

- A share for central configuration files of packages, if this functionality is used by the Scripting Framework.
- A user account which has the "Read" permissions on the share with the configuration files.

2 Available Functions

2.1 SF-AppvInstall

Overview:
Installation of a Microsoft App-V package.

Description:
Prerequisite for this function is the Microsoft App-V Client.

Syntax:
```
SF-AppvInstall "Path" "DynamicDeploymentConfiguration" "DynamicUserConfiguration" -Switches
```

Available Switches:

-Global	Publish the App-V package as global
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Installs an App-V package without a dynamic deployment configuration (xml)
SF-AppvInstall "%_PkgSource%\Setup\Example.appv" "" ""

# Installs an App-V package with a dynamic deployment configuration (xml)
SF-AppvInstall "%_PkgSource%\Setup\Example.appv" "%_PkgSource%\Setup\DynamicDeploymentConfiguration.xml" ""
```

2.2 SF-AppvUninstall

Overview:
Uninstalling an App-V package.

Description:
Prerequisite for this function is the Microsoft App-V Client.

Syntax:

```
SF-AppvUninstall "PackageId" -Switches
```

Available Switches:

-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:

```
# Uninstall an App-V package
SF-AppvUninstall "03c5ed5d-3c1c-4518-92c9-d1175c5454fb"
```

2.3 SF-AppxInstall

Overview:
Install an app package (.appx).

Description:
Adds an app package (.appx) that will install for each new user to a Windows image.

Syntax:

```
SF-AppxInstall "PackagePath" "DependencyPackagePath" -Switches
```

Available Switches:

-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:

```
# Installs an AppX package
SF-AppXInstall "%_PkgSource%\Setup\Microsoft.Whiteboard_20.10615.5289.0_x64__8wekyb3d8bbwe.appx"

# Installs an AppX package with Dependency Packages
SF-AppXInstall "%_PkgSource%\Setup\E046963F.LenovoCompanion_10.2009.18.0_neutral___k1h2ywk1493x8.AppxBundle" -
DependencyPackagePath "%_PkgSource%\Setup\Microsoft.NET.Native.Framework.1.3_1.3.24211.0_x64__8wekyb3d8bbwe.Appx",
"%_PkgSource%\Setup\Microsoft.NET.Native.Framework.1.3_1.3.24211.0_x86__8wekyb3d8bbwe.Appx",
"%_PkgSource%\Setup\Microsoft.NET.Native.Runtime.1.4_1.4.24201.0_x64__8wekyb3d8bbwe.Appx",
"%_PkgSource%\Setup\Microsoft.NET.Native.Runtime.1.4_1.4.24201.0_x86__8wekyb3d8bbwe.Appx"
```

2.4 SF-AppxUninstall

Overview:
Removes an app package (.appx).

Description:
Removes app packages (.appx) from a Windows image.

Syntax:

```
SF-AppxUninstall "DisplayName" -Switches
```

Available Switches:

-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:

```
# Uninstalls an AppX package
SF-AppxUninstall "Microsoft.Whiteboard"
```

2.5 SF-CD

Overview:
Change directory within PowerShell.

Description:
None

Syntax:
```
SF-CD "Path"
```

Available Switches:

None	

Example:
```
# Change directory
SF-CD "%_ProgramFiles32%"
```

2.6 SF-Copy

Overview:
Copying files and folders.

Description:
The data which are in use will be automatically replaced by a Windows API with the next reboot. The function uses RoboCopy in the background, the corresponding log file of the last copy operation is located in the Scripting Framework Log folder.

Syntax:

```
SF-Copy "Source" "Destination" –Switches
```

Available Switches:

-Retries	Specifies the number of retries on failed copies. Default is 3.
-RetriesTime x	Specifies the wait time between retries, in seconds. Default is 10.
-Changed	Overrides all files, whether new or old.
-LocalUserProfiles	Copy a file to all local user profiles
-Newer	Overrides only the files which are new.
-NoErrors	Error messages are ignored and the script continues.
-SelfRegister	Use this action to self-register all copied .OCX and .DLL files.
-UseVersion	Use the date version instead of the date, if the attribute is available and is not 0.0.0.0.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:

```
# Copy file
SF-Copy "%_Windows%\Notepad.exe" "%_SystemDrive%\Example" -Newer

# Copy file (with versions comparison)
SF-Copy "%_PkgSource%\Setup\WindowsSystem32" "%_WindowsSystem32%" -Newer -UseVersion

# Copy folder (changed)
SF-Copy "%_PkgSource%\Setup\ProgramFiles32" "%_ProgramFiles32%" -Changed

# Copy folder only for a 32-bit OS
SF-Copy "%_PkgSource%\Setup\ProgramFiles32" "%_ProgramFiles32%" -Changed –x86
```

```
# Copy the file Example.txt to all local user profiles
SF-Copy "%_PkgSource%\Setup\Example.txt" "AppData\Roaming" -Changed -LocalUserProfiles
```

CAUTION: A rename of a file is not possible within this function, in this case use the function SF-Rename. Not working example:

```
SF-Copy "%_Windows%\Notepad.exe" "%_SystemDrive%\Example\New.exe" -Newer
```

2.7 SF-Crypt

Overview:

Encrypting a string.

Description:

To decrypt the encrypted string, the SF-Decrypt command is available.

Syntax:

```
SF-Crypt "String"
```

Available Switches:

None	

Examples:

```
# Stores the crypted string in the PowerShell variable $CryptString
$CryptString = SF-Crypt "SecurePassword"
```

Warning: The SF-Crypt and SF-Decrypt functions are intended to crypt strings such as serial numbers so that such information is not immediately visible as plain text in the package. However, for security reasons, do not encrypt user passwords, etc., as these strings can end up being output as clear text via PowerShell.

2.8 SF-Decrypt

Overview:
Decrypts a string.

Description:
Decrypts a string that has been encrypted with SF-Crypt.

Syntax:
```
SF-Decrypt "String"
```

If the PowerShell variable is called $DecryptString, so the decrypted string is visible only with asterisks (*******) in the log file.

Caution: The variable must be called with $DecryptString, otherwise the password is displayed in plain text!

Available Switches:

None	

Examples:
```
# Decrypt the string and stores it in a PowerShell variable.
$DecryptString = SF-Decrypt "VtDlxXqGsib8oZ6ErSFgJw=="
$DecryptString2 = SF-Decrypt "VtDlxXqGsib8oZ6ErSFgJw=="
$DecryptString3 = SF-Decrypt "VtDlxXqGsib8oZ6ErSFgJw=="
$DecryptString4 = SF-Decrypt "VtDlxXqGsib8oZ6ErSFgJw=="
$DecryptString5 = SF-Decrypt "VtDlxXqGsib8oZ6ErSFgJw=="

# Decrypt the string, if input is a Scripting Framework variable
$DecryptString = SF-Decrypt (SF-Variables "%ExamplePassword%")

# Create the local user ScriptingFramework with the password in $DecryptString
SF-Run "%_WindowsSystem32%\net.exe" "user ""ScriptingFramework"" $DecryptString /Add" -Wait

# Write textfile with password and open in notepad
SF-Textfile "%_Temp%\Decrypt.txt" "$DecryptString" -Add -IfMissing
SF-Run '%_Windows%\Notepad.exe' '"%_Temp%\Decrypt.txt"' -Show
```

Warning: The SF-Crypt and SF-Decrypt functions are intended to crypt strings such as serial numbers so that such information is not immediately visible as plain text in the package. However, for security reasons, do not encrypt user passwords, etc., as these strings can end up being output as clear text via PowerShell.

2.9 SF-Delete

Overview:
Deleting a file.

Description:
Optionally the open handles will be closed automatically.

Syntax:
```
SF-Delete "Path" -Switches
```

Available Switches:

-DisableTaskkill	There is no automatic taskkill for the process.
-LocalUserProfiles	Deletes a file recursively from all local user profiles (see examples)
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Remove file
SF-Delete "%_SystemDrive%\Example\File.txt"

# Remove example.log from all user profiles
SF-Delete "AppData\Roaming\Example.txt" -LocalUserProfiles

Result Example (LocalUserProfiles):
20:03:2017-08:52:57    INFO: ===== Delete ================================================
20:03:2017-08:52:57    INFO:      Path: C:\Users\Public\AppData\Roaming\Example.txt
20:03:2017-08:52:57    INFO:      Path: C:\Users\Username1\AppData\Roaming\Example.txt
20:03:2017-08:52:57    INFO:      Path: C:\Users\Username2\AppData\Roaming\Example.txt
20:03:2017-08:52:57    INFO:      Path: C:\Users\Username3\AppData\Roaming\Example.txt
```

2.10 SF-DriverInstall

Overview:
Installs a device driver.

Description:
Installs a single device driver or all device drivers from a folder including subfolders.

Syntax:
```
SF-DriverInstall "Path" -Switches
```

Available Switches:

-NoErrors	The exit codes are ignored and it will continue with the script
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Installs a single driver
SF-DriverInstall "%_PkgSource%\Setup\Driver\Sp40_7_x64.inf"

# Installs all device drivers from a folder including subfolders
SF-DriverInstall "%_PkgSource%\Setup\Drivers"
```

2.11 SF-DriverUninstall

Overview:
Uninstalls a device driver.

Description:
Uninstalls a driver from the Windows Driver Store.

Syntax:
```
SF-DriverUninstall "Path" "SearchString" -Switches
```

Available Switches:

-NoErrors	The exit codes are ignored and it will continue with the script
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Uninstalls a specific driver from the DriverStore (oem*.inf), if the search string was found
SF-DriverUninstall "%_Windows%\INF" "MPC3504/C3004E-23C/MPC6004/C5504/C4504E-23C PCL 6 Printer Driver"

# Uninstalls a single driver by path
SF-DriverUninstall "%_Windows%\INF\oem43.inf"

# Uninstalls a single driver by name
SF-DriverUninstall "oem43.inf"
```

2.12 SF-Exit

Overview:
For premature exit of a packet (error).

Description:
Creates a corresponding Scripting Framework log entry and then exits the script with the default return code 1603. Custom return codes are also possible.

Syntax:
```
SF-Exit "Message" "ReturnCode" –Switches
```

Available Switches:

-Severity	Defines the severity of the message. Valid values are: Custom, Error, Info, Warning and Title
-ShowBalloonTip	Displays a balloon tip in the lower right corner of the screen. The switch must be used in the current user context (no BalloonTip is displayed if the command is executed in another context).

Examples:
```
# Sample 1 – Exit Scripting Framework with 1603
SF-Exit "My Example Message"

# Sample 2 - Exit Scripting Framework (Check Condition)
SF-LoadVariables "%c_ConfigPath%\%_PkgCFGName%.cfg" "CommonClientSettings"
If ((SF-Variables "%_VarPrefix-Example%") -eq "") {SF-Exit "Value for Variable <Example> must be set in %_PkgCFGName%.cfg"}

# Sample 3 - Exit Scripting Framework with a custom return code
SF-Exit "My Example Message" "1610" -Severity Warning
```

2.13 SF-ExtractArchive

Overview:
Extracts a zip file.

Description:
Extracts a zip archive to the specified destination.

Syntax:
```
SF-ExtractArchive "Path" "Destination" -Switches
```

Available Switches:

-CacheLocal	Before the unpacking starts it will start a automatically a check, if the file is on a network or on a local drive. Should it not be a local drive, the file will be automatically locally cached and unzipped.
-Log	Unpacking is displayed in detail in the Scripting Framework Log.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Extract archive file
SF-ExtractArchive "%_PkgSource%\Setup\ProgramFiles32.zip" "%_ProgramFiles32%\Example"

# Extract archive file
SF-ExtractArchive "%_PkgSource%\Setup\ProgramFiles64.zip" "%_ProgramFiles64%\Example" -x64 -CacheLocal
```

2.14 SF-File

Overview:
Checks whether a file or folder exists.

Description:
The function returns the value true or false.

Syntax:

```
SF-File "Path"
```

Available Switches:

None	

Examples:

```
# Check file
If ((SF-File "%_Windows%\Notepad.exe") -eq $True) {
    SF-Log "File found"
    SF-Run "%_Windows%\Notepad.exe" -Show
}
Else {
    SF-Log "File not found"
}

# Check folder
If ((SF-File "%_Windows%") -eq $True) {
    SF-Log "Folder found"
}
```

2.15 SF-FileSearch

Overview:
Search for a file.

Description:
The function looks for a file and returns the result as an array.

Syntax:
```
SF-FileSearch "Path" "Filename"
```

Available Switches:

-Depth x	Control the depth of recursion.

Examples:
```
# Example 1: Search the drive C:\ for the file Example.txt and write the path of the found files into the log
$SearchResult = SF-FileSearch "%_SystemDrive%" "Example.txt"

ForEach ($Result in $SearchResult) {
    SF-Log $Result
}

# Example 2: Search the drive C:\ for the file Example.txt and write the path of the found files into the log - With the
maximum folder depth of 2
$SearchResult = SF-FileSearch "%_SystemDrive%" "Example.txt" -Depth 2

ForEach ($Result in $SearchResult) {
    SF-Log $Result
}
```

2.16 SF-FileVersion

Overview:
Reads the version number of a file and returns it. The function can be combined with the PowerShell comparison operators (-eq, -ne, -lt, -le, -gt and -ge) and create corresponding queries.

Description:
The function returns the value as the file version.

Syntax:
```
SF-FileVersion "Path" -Switches
```

Available Switches:

-AsString	Specifies the version with the data type [string] instead of [Version].

Examples:
```
# Checks the file version of notepad
If ((SF-FileVersion "%_Windows%\Notepad.exe") -gt "6.1.7600.16300") {
    SF-Run "%_Windows%\Notepad.exe" -Show
}
Else {
    SF-Log "Version is less than indicated" -Severity Error
}
```

2.17 SF-FontInstall

Overview:
Installs a single font or all in one folder.

Description:
The following types are supported (.from, .fnt, .ttf, .ttc, .otf, .pfm).

Syntax:

```
SF-FontInstall "Path"
```

Available Switches:

-CurrentSessionLoad	With this switch the font is only temporarily loaded for one user. After logoff, the font is no longer available. This switch can be used on terminal servers to manage licensed fonts. Normally, the function in the Scripting Framework "Scripting Mode" is used together with a Share and activated Access Based Enumeration. For questions, the Windows Client Management AG is at your disposal.

Examples:

```
# Install a single font
SF-FontInstall "%_PkgSource%\Setup\Example.ttf"

# Install all fonts in folder
SF-FontInstall "%_PkgSource%\Setup"
```

2.18 SF-FontUninstall

Overview:
Uninstalling a font.

Description:
The font is uninstalled from the file name. Instead of the file name, it is possible to specify a folder and all fonts in this folder will be uninstalled.

Syntax:

```
SF-FontUninstall "Path"
```

Available Switches:

None	

Examples:

```
# Uninstall a single font
SF-FontUninstall "Example.ttf"

# Uninstall all fonts in folder
SF-FontUninstall "%_PkgSource%\Setup"
```

2.19 SF-GroupMembership

Overview:
Check whether a username is a member of a specific Active Directory group, or if a group include a specified user.

Description:
The function returns the value true or false. By a function call without the group name, the array $GroupMembership is created, which can be used for further queries. The array can diverse queries, that can be realized without multiple AD access. See Examples:

Syntax:

```
SF-GroupMembership "Object" "MemberOf"
```

Available Switches:

None	

Examples:

```
# Check if the user ExampleUser a member from the group ExampleGroup
# Variables can be used such as %_User%
If ((SF-GroupMembership "ExampleUser" "ExampleGroup") -eq $True) {
    SF-Log "The user is a member"
}

# Create the array $GroupMembership and check for group membership
$GroupMembership = SF-GroupMembership "%_User%"

If ($GroupMembership -contains 'ExampleGroup') {
    SF-Log "The user is a member of ExampleGroup"
}
ElseIf ($GroupMembership -contains 'ExampleGroup2') {
    SF-Log "The user is a member of ExampleGroup2"
}
Else {
    SF-Log "The user is not a member"
}
```

2.20 SF-INIRead

Overview:
Reading from a file in INI format.

Description:
The read value is stored accordingly in the Registry as Scripting Framework variable.

Syntax:

```
SF-INIRead "Path" "Section" "Name" "VariableName"
```

Available Switches:

-Machine	Saves the machine-related variable under HKLM\Software\Scripting Framework\Variables
-Script	Creates a temporary PowerShell Variable
-User	Saves the user-related variable under HKCU\Software\Scripting Framework\Variables
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
ExampleFile.ini
[Section]
Name=Example

```
# Example 1 - Generate the Scripting Framework variable INIRead_Var with the value „Example“
SF-INIRead "%_SystemDrive%\Example\ExampleFile.ini" "Section" "Name" "INIRead_Var"


# Example 2 - Read INI File and Check the Value
SF-INIRead "%_Windows%\win.ini" "Mail" "CMCDLLNAME32" "New_IniReadVar"
If ((SF-Variables "%New_IniReadVar%") -ne "mapi32.dll") {
    SF-Exit "Error occures, the value is not equal mapi32.dll"
}
```

2.21 SF-INIWrite

Overview:
Create, edit and delete values in a file in INI format.

Description:
None

Syntax:
```
SF-INIWrite "Path" "Section" "Name" "Value"
```

Available Switches:

-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Write a value to the file ExampleFile.ini
SF-INIWrite "%_SystemDrive%\Example\ExampleFile.ini" "Section" "Name" "Example"
```

Output ExampleFile.ini:
[Section]
Name=Example

```
# Delete value
SF-INIWrite "%_SystemDrive%\Example\ExampleFile.ini" "Section" "Name" "{nil}"

# Delete section
SF-INIWrite "%_SystemDrive%\Example\ExampleFile.ini" "Section" "{nil}" "{nil}"
```

2.22 SF-Installed

Overview:
Checks if a software is installed on the computer.

Description:
A DisplayName or MSI ProductCode can be used for the search. The search for an SF Package Identifier is also supported. The function returns true if the value is found otherwise false.

Syntax:

```
SF-Installed "Product"
```

Available Switches:

None	

Examples:

```
#  Checks if the product is installed with the specified MSI ProductCode
If ((SF-Installed "{02382870-19C7-3ACD-BBAE-F6E3760947DC}") -eq $True) {
    SF-Log "The product is installed"
}

# Checks if the product is installed with the specified DisplayName
If ((SF-Installed "Citrix Receiver 4.6") -eq $True) {
    SF-Log "The product is installed"
}


# Checks if a product with the DisplayName like Citrix is installed (Using Wildcards)
If ((SF-Installed "Citrix*") -eq $True) {
    SF-Log "The product is installed"
}


# Check for a Scripting Framework package
If ((SF-Installed "Free_PDF_Writer_414_MUI_UNV_01") -eq $True)
{
    SF-Log "The package is installed"
}
```

2.23 SF-Language

Overview:
For the determination of the language within a MUI pack.

Description:
The function returns the appropriate language code returned in the variable *_PkgLang*, which can be subsequently used for the installation or configuration.

Language codes Examples:

Language	Code
German	1031
English	1033
French	1036
Italian	1040

Case 1 - Use the function in a package:
When the function is checked first, it's carried out in which context the installation is. When installing (Install.ps1, Uninstall.ps1), the default language of the client (OS) is set. Subsequently, the examination takes place, whether it is an available language. If the read OS language matches the available languages, the variable *_PkgLang* is created with the corresponding value. Should the client language does not match a supported language, the fallback language is used.

By call to the function in UserScript "Install User.ps1", is determined the current language of the current user instead of the default language of the client (OS).

Case 2 - Overriding the language via Command Line Parameters:
The function can be overrided by a command line parameter over the install.exe from the packet:

```
Install.exe -StartLanguage "1031"
```

In this case the function *SF-Language* does not carry detection of the language out and in the variable _PkgLang is automatically passed as a command line parameter language code. A further review will not take place.

Case 3 - language is fixed by configuration:
If it's not to check the default language, it is possible to fix the language in the registry entry. Thus, in each case to determine whether the fixed language is available, otherwise the *FallbackLanguage* is used. The configuration is typically used only in conjunction with an appointment server. Configuration:

Key: HKEY_LOCAL_MACHINE\SOFTWARE*Wow6432Node*\ScriptingFramework\Config
Name: ForceMachineLanguage
Value: 1031
Type: REG_SZ

Syntax:

```
SF-Language "AvailableLanguages" "FallbackLanguage"
```

Available Switches:

None	

Examples:

```
# Language (Example 1)
SF-Language "1031,1033,1036,1040" "1033"

# Installation Reader
SF-MSIInstall "%_PkgSource%\Setup\AcroRead.msi" "TRANSFORMS='%_PkgSource%\Setup\%_PkgLang%.mst' ALLUSERS='1'
EULA_ACCEPT='YES' SUPPRESS_APP_LAUNCH='YES' /qn /norestart"


# Language (Example 2)
SF-Language "1031,1033,1036" "1033"

# DEU
If ((SF-Variables "%_PkgLang%") -eq "1031") {
   SF-Register32 "HKey_Local_Machine" "Software\Example" "Language" "de-DE" "REG_SZ"
}

 # ENU
ElseIf ((SF-Variables "%_PkgLang%") -eq "1033") {
    SF-Register32 "HKey_Local_Machine" "Software\Example" "Language" "en-US" "REG_SZ"
 }

 # FRA
ElseIf ((SF-Variables "%_PkgLang%") -eq "1036") {
   SF-Register32 "HKey_Local_Machine" "Software\Example" "Language" "fr-FR" "REG_SZ"
}
```

2.24 SF-LoadVariables

Overview:
Loading Scripting Framework variables from a file in INI format.

Description:
The loaded variables are stored in the Registry as Scripting Framework variables and can be used subsequently in the script. The CFG file is searched automatically in two stages, for details see Loading of package variables for dynamic packages.

Syntax:

```
SF-LoadVariables "Path" "Section" -Switches
```

Available Switches:

-NoErrors	Error messages are ignored and the script continues. This switch is usually used in "Scripting Mode". Thus, the script continues to run if the client should not be located on the corporate network.
-NoVarPrefix	By default is automatically added to the loaded variable name a prefix a_PkgIdentifier-VariableName. This configuration allows the automatic addition of a prefix (information from the Package.xml) stop.
-ShowBalloonTip	Displays a balloon tip in the lower right corner of the screen, if an error occurs. The switch must be used in the current user context (no BalloonTip is displayed if the command is executed in another context). This switch is usually only used in Scripting Mode.
-User	Load the variables in the user context (HKCU).

Examples:
Inhalt der CFG Datei:

```
; === Client Settings ================================================
[CommonClientSettings]
ServerLocation=https://xen.citrix.net
```

```
# Load Variables
SF-LoadVariables "%c_ConfigPath%\%_PkgCFGName%.cfg" "CommonClientSettings"
```

In the registry, the charged Scripting Framework variables from the CFG are automatically stored with the corresponding prefix (a_ + Identifier from the Package.xml) (the a_ stands for Application):
a_Citrix_Receiver_Enterprise_13_MUI_UNV-ServerLocation

To use the variable "Server Location" without the full variable name including prefix, the following variant can be used to access:

```
SF-Run "%_PkgSource%\Setup\CitrixReceiverEnterprise.exe" "/silent /noreboot SERVER_LOCATION=%_VarPrefix-ServerLocation%" -Wait
```

By using _VarPrefix- are an adaptation in Package.xml, no changes in the script needed (this is the recommended option) in the case. However, the response is also possible through the complete variable name:

```
SF-Run "%_PkgSource%\Setup\CitrixReceiverEnterprise.exe" "/silent /noreboot SERVER_LOCATION=%a_Citrix_Receiver_Enterprise_13_MUI_UNV-ServerLocation%" -Wait
```

2.25 SF-Lockscreen

Overview:

Locks the login user screen during installation (this works also under the System account).

Description:

The user's screen is locked, so user input can be prevented. The lock can be canceled by entering a password: Use the mouse to click in the Lockscreen and enter the password on the keyboard. The default password is unlock, unless it was passed by another parameter.

Syntax:

```
SF-Lockscreen "Title" "Description" "Logo" -Switches
```

Available Switches:

-Timeout x	Timeout in seconds for the automatic repeal of Locks. Without this parameter, no timeout is set (value 0). Without the DisableAutoUnlock parameters of the lock, it will be released at the end of the script, even if the timeout was not reached.
-DisableAutoUnlock	After successful installation of the package, the Lockscreen is canceled by default. This switch allows the automatic cancellation of Locks suppress. The Lockscreen remains active until it is canceled with the Stop switch.
-DisableErrorUnlock	If Scripting Framework found an error within the script notes (1603), the Lockscreen will be canceled by default. With this switch the Lockscreen remain active when an error is detected.
-Password x	The Default Password (unlock) will be overridden by this parameter.
-Permanent	The lockscreen is registered in the registry, so that it is started again at a user login after a reboot. This remains active until it is canceled with the Stop switch.
-Stop	Unprotecting for all users. If the switch permanent was used, the automatic start is removed from the registry.

Examples:

```
# Lockscreen without logo and timeout and with default
SF-Lockscreen "Example" "This is an example"

# Lockscreen with two lines and a custom unlock password. Use `n for a new line
SF-Lockscreen "Example" "Line1 `nLine2" -Password "secret"

# Lockscreen without logo and timeout and with a custom unlock password
SF-Lockscreen "Example" "This is an example" -DisableAutoUnlock -Timeout 10 -Password "secret"

# Lockscreen with logo, disable auto unlock and disable automatic unlock on error
SF-Lockscreen "Example" "This is an example" "%_PkgSource%\Setup\Logo.jpg" -DisableAutoUnlock -DisableErrorUnlock

# Stops the Lockscreen
SF-Lockscreen -Stop
```

Lockscreen with logo:

2.26 SF-Log

Overview:
Writes an entry in the log file, by default the Severity Output is CUSTOM.

Description:
None

Syntax:
```
SF-Log "Message" -Switches
```

Available Switches:

-Severity	Defines the severity of the message. Valid values are: Custom, Error, Info, Warning and Title

Examples:
```
# Write log (CUSTOM)
SF-Log "Example log entry"

# Write log (Info)
SF-Log "ProgramFiles32 folder: %_ProgramFiles32%" -Severity Info

# Write log (ERROR)
SF-Log "Example log entry" -Severity Error
```

2.27 SF-Logoff

Overview:
Log off of all or a specific Windows user.

Description:
None

Syntax:
```
SF-Logoff "User" -Switches
```

Available Switches:

-Delay x	Delays the logoff by x seconds.

Examples:
```
# Log out all logged in users
SF-Logoff

# Log out all logged in users with a delay of 10 seconds
SF-Logoff -Delay "10"

# Log off a specific user
SF-Logoff "User1"
```

2.28 SF-MD

Overview:
Created an empty folder, including subfolders.

Description:
None

Syntax:
```
SF-MD "Path" -Switches
```

Available Switches:

-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Create folder
SF-MD "%_SystemDrive%\Example\Subfolder"
```

2.29 SF-MessageBox

Overview:
Displays a message box with Windows Forms for the logged-in user. This also works when run under the SYSTEM account context.

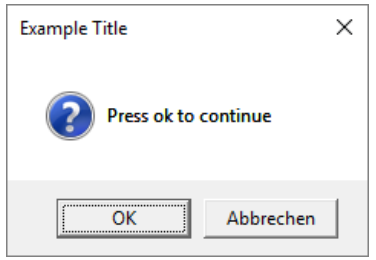
Description:
The function displays a custom MessageBox. The parameters are used to define which icons and buttons are to be used. The return value is the clicked button, the possible return values are Abort, Cancel, Ignore, No, OK, Retry or Yes.

Syntax:
```
SF-MessageBox "Title" "Text" -Switches
```

Verfügbare Switches:

-Button	Defines which buttons are displayed. Valid values are: AbortRetryIgnore, OK, OKCancel, RetryCancel, YesNo und YesNoCancel
-Icon	Defines which icon is displayed. Valid values are: Information, Asterisk, Question, Stop und Warning
-Wait	Wait until the user clicks a button. Attention: Without the Wait parameter the result can not be evaluated, in this case it is always OK!

Examples:
```
# This example creates a msgbox with the Ok and Cancel button and the Question Icon. Once the message box is displayed it creates the Result variable with the message box selection choosen. Once the message box is done you can use an if statement to finish the script.

$Result = SF-MessageBox "Example" "Press ok to continue" -Button OKCancel -Icon Question -Wait

If ($Result -eq "OK" )
{
	SF-Log "The user has pressed 'OK'"
}
ElseIf ($Result -eq "Cancel")
{
	SF-Log "The user has pressed 'Cancel'"
}
```



2.30 SF-MSIGetProperty

Overview:
Reads the value of a property from an MSI file.

Description:
The command reads the property from the specified MSI file and returns the value.

Syntax:
```
SF-MSIGetProperty "Path" "PropertyName" -Switches
```

Available Switches:

-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Reads the property ARPHELPLINK and returns it.
$Value = SF-MSIGetProperty "%_PkgSource%\Setup\Google Earth.msi" "ARPHELPLINK"
SF-Log $Value -Severity Info
```

2.31 SF-MSIInstall

Overview:
Installs an MSI file and automatically evaluates the return code (0, 1605, 1641 and 3010).

Description:
In the case that during the installation of the MSI, a reboot was detected, Scripting Framework terminated at the end of the script with the return code 3010. If the product is already installed, the function will automatically perform a repair with the parameter / foums, except the Switch "RemoveFirstOnRepair" is used.

Syntax:
```
SF-MSIInstall "Path" "Arguments" -Switches
```

Available Switches:

-Log	Write the MSI log into the folder "C:\Windows\Scripting Framework\Logs\Software".
-NoErrors	The MSI exit codes are ignored and it will continue with the script.
-RemoveFirstOnRepair	If the product is already installed on your system, will run, instead of a repair, a Remove first and then the installation.
-SkipIfInstalled	The installation is skipped if the program is already installed.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# MSI installation (x86 and x64)
SF-MSIInstall "%_PkgSource%\Setup\RemotelyAnywhereNH.msi" "ALLUSERS='1' /qn /norestart"

# MSI installation (only x86 and with log)
SF-MSIInstall "%_PkgSource%\Setup\RemotelyAnywhereNH.msi" "ALLUSERS='1' /qn /norestart" -x86 -Log

# MSI installation (only x86, with log, and remove first if already installed)
SF-MSIInstall "%_PkgSource%\Setup\RemotelyAnywhereNH.msi" "ALLUSERS='1' /qn /norestart" -x86 -Log -RemoveFirstonRepair
```

2.32 SF-MSIPatchInstall

Overview:
Installs an MSI patch file (.msp) and automatically evaluates the return code (0, 1605, 1641, 1642 and 3010).

Description:
None

Syntax:
```
SF-MSIPatchInstall "Path" "Arguments" -Switches
```

Available Switches:

-Log	Write the MSI log into the folder "C:\Windows\Scripting Framework\Logs\Software".
-NoErrors	Error messages are ignored and the script continues.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Install a MSI patch
SF-MSIPatchInstall "%_PkgSource%\Setup\Patch.msp" "/qn /norestart"

# Install a MSI patch (only x86 and write log)
SF-MSIPatchInstall "%_PkgSource%\Setup\Patch.msp" "/qn /norestart" -x86 -Log

# Installs all MSI patches from the specified folder
SF-MSIPatchInstall "%_PkgSource%\Setup" "/qn /norestart"
```

2.33 SF-MSIRepair

Overview:
Running a MSI repair.

Description:
The function is used for installations which have a bootstrapper (eg setup.exe) to install, however, support a MSI repair. The SF-MSIInstall command runs automatically when you exist the application of a repair, whereby the SF-MSIRepair command is not needed in a pure MSI installation without bootstrapper.

Syntax:
```
SF-MSIRepair "ProductCode" "Arguments" -Switches
```

Available Switches:

-Log	Write the MSI log into the folder "C:\Windows\Scripting Framework\Logs\Software".
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Repair a MSI installation
SF-MSIRepair "{F27DC4B4-42BC-43AF-BF5F-18ED9A7867BD}" "/qn /norestart"
```

2.34 SF-MSIUninstall

Overview:
Uninstall an MSI product and automatically evaluates the return code (0, 1605, 1641 and 3010).

Description:
None

Syntax:
```
SF-MSIUninstall "ProduktCodeOrFile" "Arguments" -Switches
```

Available Switches:

-NoErrors	The MSI exit codes are ignored and it will continue with the script.
-Log	Write the MSI log into the folder "C:\Windows\Scripting Framework\Logs\Software".
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# MSI uninstall (ProductCode)
SF-MSIUninstall "{F27DC4B4-42BC-43AF-BF5F-18ED9A7867BD}" "/qn /norestart"

# MSI Uninstall (MSI file)
SF-MSIUninstall "%_PkgSource%\Setup\RemotelyAnywhereNH.msi" "/qn /norestart"
```

2.35 SF-MSIUninstallByDisplayName

Overview:
Uninstalling an application by name.

Description:
The function searches for the specified display name (recursively) and uninstall the programs that correspond to this name. This requires an MSI installation, other setup routines are not supported. A search with wildcards is also possible.

Warning: The command can cause danger when used properly, not hazards and may under certain circumstances with a "false" display name uninstall unwanted applications! Therefore this command should be used with caution.

Syntax:
```
SF-MSIUninstallByDisplayName "DisplayName" "Arguments" -Switches
```

Available Switches:

-Log	Write the MSI log into the folder "C:\Windows\Scripting Framework\Logs\Software".
-NoErrors	The MSI exit codes are ignored and it will continue with the script.

Examples:
```
# Uninstall
SF-MSIUninstallByDisplayName "Microsoft Visual C++ 2008"

# Uninstall with wildcards
SF-MSIUninstallByDisplayName "*Microsoft Visual*"

# By default the parameters /qn /norestart are given, these can be customized if properties are needed
SF-MSIUninstallByDisplayName "*Microsoft Visual*" "REMOVESETTINGS=true /qn /norestart"
```

2.36 SF-MSIXInstall

Overview:
Installs an MSIX package.

Description:
The command installs an MSIX package or MSIX bundle.

Syntax:
```
SF-MSIXInstall "Path" "DependencyPackagePath" -Switches
```

Available Switches:

-NoErrors	If errors occur during installation, they are ignored.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Example:
```
# MSIX installation
SF-MSIXInstall "%_PkgSource%\Setup\MSIX.Commander_1.0.7.2-x64.msix"

# Installs an MSIX package with Dependency Packages
SF-MSIXInstall "%_PkgSource%\Setup\Barcode_1.2.18.0_x64.msixbundle" -DependencyPackagePath
"%_PkgSource%\Setup\Dependencies\x64\Microsoft.NET.Native.Framework.2.2.appx","%_PkgSource%\Setup\Dependencies\x64\Microsoft.NET.Native.Runtime.2.2.appx","%_PkgSource%\Setup\Dependencies\x64\Microsoft.VCLibs.x64.14.00.appx"
```

2.37 SF-MSIXUninstall

Overview:
Uninstall an MSIX package.

Description:
The command uninstalls an MSIX package by Name or PackageFullName.

Syntax:
```
SF-MSIXUninstall "Package" -Switches
```

Available Switches:

-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# MSIX uninstallation by Name
SF-MSIXUninstall "PascalBerger.MSIXCommander"

# MSIX uninstallation by PackageFullName
SF-MSIXUninstall "PascalBerger.MSIXCommander_1.0.7.2_x64__ajjbhed1xnq88"

# MSIX uninstallation by PackageFullName with wildcard
SF-MSIXUninstall "PascalBerger.MSIXCommander_1.0.7.2_*"
```

2.38 SF-NTFSPerm

Overview:
Set NTFS permissions on a file or a folder.

Description:
The three permission type Full, Modify and Read are possible.

Syntax:
```
SF-NTFSPerm "Path" "User" "Permission" -Switches
```

Available Switches:

-InheritsFromParent	Inherited permissions from the parent object, which changes will be reset in the subfolders.
-Remove	Removes the appropriate user from the permissions.

Examples:
```
# Set full permission for the users group
SF-NTFSPerm "%_ProgramFiles32%\Example\Example.ini" "%_GroupUsers%" "Full"

# Set modify permission for the users group
SF-NTFSPerm "%_ProgramFiles32%\Example" "%_GroupUsers%" "Modify"

# Set read permission for the users group
SF-NTFSPerm "%_ProgramFiles32%\Example" "%_GroupUsers%" "Read"

# Remove the users group from permissions
SF-NTFSPerm "%_ProgramFiles32%\Example" "%_GroupUsers%" -Remove
```

2.39 SF-NuGetInstall

Overview:
Installs a NuGet software package.

Description:
Installs a NuGet package via the PackageManagement Module.

Syntax:

```
SF-NuGetInstall "Name" -Switches
```

Available Switches:

-Detailsd	Displays detailed information about the operation done by the command.
-SkipDependencies	Skips the installation of software dependencies.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:

```
# Installs an NuGet package from a local source
SF-NuGetInstall "%_PkgSource%\Setup\notepad2.4.2.25.3.nupkg"

# Installs a Nuget Package via the NuGet PackageProvider (Internet)
SF-NuGetUninstall "notepad2"
```

2.40 SF-NuGetUninstall

Overview:
Uninstalls a NuGet software package.

Description:
Uninstall a NuGet package via the PackageManagement Module.

Syntax:

```
SF-NuGetUninstall "Name" -Switches
```

Available Switches:

-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:

```
# Uninstalls the NuGet package named notepad2
SF-NuGetUninstall "notepad2"
```

2.41 SF-Path

Overview:
Extends the Environment Variable Path to the specified path.

Description:
None

Syntax:
```
SF-Path "Path" -Switches
```

Available Switches:

-AtFirst	Inserts the path at the beginning
-Force	If the path entry in the interaction with AtFirst already exists, the entry is moved to the first position
-Remove	Removes the path from the environment variable Path
-User	Sets or removes the path for the current UserContext

Examples:
```
# Add a path
SF-Path "%_SystemDrive%\Example"

# Remove a path
SF-Path "%_SystemDrive%\Example" -Remove
```

2.42 SF-PinnedApplication

Overview:
This function are used to pin and unpin programs from the taskbar and Start menu.

Description:
The feature requires Windows Vista or higher.

Syntax:

```
SF-PinnedApplication "Path" -Switches
```

Available Switches:

-PinToTaskbar	Pin an application to the Taskbar
-UnPinFromTaskbar	Unpin an application from the Taskbar
-PinToStartMenu	Pin an application to the Start menu
-UnPinFromStartMenu	Unpin an application to the Start menu

Examples:

```
# Pin to taskbar
SF-PinnedApplication "%_Windows%\explorer.exe" -PinToTaskbar

# Unpin from taskbar
SF-PinnedApplication "%_Windows%\explorer.exe" -UnPinFromTaskbar

# Pin to startmenu
SF-PinnedApplication "%_CommonPrograms%\Accessories\WordPad.lnk" -PinToStartMenu

# Unpin from startmenu
SF-PinnedApplication "%_CommonPrograms%\Accessories\WordPad.lnk" -UnPinFromStartMenu
```

2.43 SF-ProgramsAndFeatures

Overview:
Creates an entry under "Programs and Features". By default, the values from the Package.xml are used unless a parameter is specified.

Description:
Create an uninstall entry to display the software under "Programs and Features". If a package is uninstalled, this entry is automatically removed. This entry is also removed when the SF-Supersedence command is applied.

Syntax:

```
SF-ProgramsAndFeatures -Switches
```

Available Switches:

-DisplayIcon	The icon that is displayed. The scripting framework icon is used as default.
-DisplayName	Name of the application.
-DisplayVersion	Version of the application.
-Publisher	Publisher/Manufacturer of the application.
-Remove	Removes the entry.
-UninstallString	Command to uninstall the application.

Examples:

```
# Creates a standard entry from the Package.xml informations
SF-ProgramsAndFeatures

# Creates a entry with customized values
SF-ProgramsAndFeatures -DisplayIcon "%_ProgramFiles32%\CDBurnerXP\cdbxpcmd.exe,0" -DisplayName "Example Titel"

# Removes the entry
SF-ProgramsAndFeatures -Remove
```

2.44 SF-RD

Overview:
Removes a directory.

Description:
All processes and open handles of this directory are automatically closed.

Syntax:
```
SF-RD "Path" -Switches
```

Available Switches:

-DisableTaskkill	The processes in the corresponding folder are not terminated automatically.
-Empty	Deletes the folder only if it is empty.
-LocalUserProfiles	Deletes a folder recursively from all local user profiles (see examples)
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Remove folder
SF-RD "%_SystemDrive%\Example"

# Remove folder if empty
SF-RD "%_SystemDrive%\Example" -Empty

# Remove example.log from all user profiles
SF-RD "AppData\Roaming\Example" -LocalUserProfiles

Result Example (LocalUserProfiles):
20:03:2017-08:52:57    INFO: ===== RD ============================================
20:03:2017-08:52:57    INFO:      Path: C:\Users\Public\AppData\Roaming\Example
20:03:2017-08:52:57    INFO:      Path: C:\Users\Username1\AppData\Roaming\Example
20:03:2017-08:52:57    INFO:      Path: C:\Users\Username2\AppData\Roaming\Example
20:03:2017-08:52:57    INFO:      Path: C:\Users\Username3\AppData\Roaming\Example
```

2.45 SF-ReloadVariables

Overview:
The command rewrites the default variables of Scripting Framework into the registry.

Description:
Scripting Framework checks the last boot time each time it runs. The available default variables are only reloaded if the boot time is different. This guarantees a very fast loading of the engine. With this function the variables within a script can be forced.

Syntax:
```
SF-RemoveVariables -Switches
```

Available Switches:

-Machine	Only reloads the computer variables (HKLM).
-User	Only reloads the user variables (HKCU).

Examples:
```
# Reload machine and user variables
SF-ReloadVariables

# Reload only machine variables
SF-ReloadVariables -Machine

# Reload only user variables
SF-ReloadVariables -User
```

2.46 SF-RemoveVariables

Overview:
The function automatically removes the variables which have been loaded with the SF-LoadVariables function (wihout the parameter -NoVarPrefix).

Description:
The function is typically used only in Uninstall.ps1 script.

Syntax:

```
SF-RemoveVariables -Switches
```

Available Switches:

-User	Deletes variables from the user context in which the script runs.

Examples:

```
# Remove Scripting Framework variables
SF-RemoveVariables
```

2.47 SF-Register32

Overview:
Writes and deletes registry keys under the 32-bit hive.

Description:
There are the following registry types valid: REG_SZ, REG_DWORD, REG_DWORD, REG_BINARY, REG_EXPAND_SZ and REG_MULTI_SZ.

Syntax:
```
SF-Register32 "RegHive" "SubKey" "ValueName" "ValueKind" -Switches
```

Available Switches:

-Add	Advanced a REG_MULTI_SZ to the specified value. If the switch is not specified, the value is overwritten.
-Remove	Removes the specified value from a REG_MULTI_SZ key.
-ConvertToHex	For writing a string to a Binary Key. With the parameters of the string converted to the correct format.
-LocalUserProfiles	Runs the command on all available local user profiles.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Registry (32-bit)
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_String" "%_ProgramFiles32%" "REG_SZ"
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_Dword" "32" "REG_DWORD"
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_Qword" "32" "REG_QWORD"
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_EXPAND_SZ" "Expand_32" "REG_EXPAND_SZ"
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_MULTI_SZ" "Value1_32" "REG_MULTI_SZ" -Add
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_Binary" "H#32698347997fddff" "REG_BINARY"
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_Binary" "32,69,83,47,99,7f,dd,ff" "REG_BINARY"
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_Binary" "Example" "REG_BINARY" -ConvertToHex

# Writes the value to all local user profiles
SF-Register32 "HKEY_CURRENT_USER" "SOFTWARE\Example" "Hello" "World" "REG_SZ" -LocalUserProfiles

# Remove a value (32-Bit)
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "32_String" "{nil}" "REG_SZ"

# Remove a key (32-Bit)
SF-Register32 "HKey_Local_Machine" "SOFTWARE\Example" "{nil}" "{nil}" "REG_SZ"
```

2.48 SF-Register64

Overview:
Writes and deletes registry keys under the 64-bit hive.

Description:
There are the following registry types valid: REG_SZ, REG_DWORD, REG_DWORD, REG_BINARY, REG_EXPAND_SZ and REG_MULTI_SZ.

CAUTION: On a 32-bit OS no action is taken and there is no output info.

Syntax:
```
SF-Register64 "RegHive" "SubKey" "ValueName" "ValueKind" -Switches
```

Available Switches:

-Add	Advanced a REG_MULTI_SZ to the specified value. If the switch is not specified, the value is overwritten.
-Remove	Removes the specified value from a REG_MULTI_SZ key.
-ConvertToHex	For writing a string to a Binary Key. With the parameters of the string converted to the correct format.

Examples:
```
# Registry (64-bit)
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_String" "%_ProgramFiles64%" "REG_SZ"
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_Dword" "64" "REG_DWORD"
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_Qword" "64" "REG_QWORD"
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_EXPAND_SZ" "Expand_64" "REG_EXPAND_SZ"
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_MULTI_SZ" "Value1_64" "REG_MULTI_SZ" -Add
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_Binary" "H#32698347997fddff" "REG_BINARY"
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_Binary" "32,69,83,47,99,7f,dd,ff" "REG_BINARY"
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_Binary" "Example" "REG_BINARY" -ConvertToHex

# Remove a value (64-Bit)
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "64_String" "{nil}" "REG_SZ"

# Remove a key (64-Bit)
SF-Register64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "{nil}" "{nil}" "REG_SZ"
```

2.49 SF-RegisterFile32

Overview:
Register a file.

Description:
Register a 32-bit file.

Syntax:
```
SF-RegisterFile32 "Path"
```

Available Switches:

None	

Examples:
```
# Register a file (32-bit)
SF-RegisterFile32 "%_WindowsSystem32%\Example.dll"
```

2.50 SF-RegisterFile64

Übersicht:
Register a file.

Description:
Register a 64-bit file.

Syntax:
```
SF-RegisterFile64 "Path"
```

Available Switches:

None	

Examples:
```
# Register a file (64-bit)
SF-RegisterFile64 "%_WindowsSystem64%\Example.dll"
```

2.51 SF-RegPerm32

Overview:
Modifies the 32-bit registry permissions.

Description:
None

Syntax:
```
SF-RegPerm32 "RegHive" "SubKey" "User" -Switches
```

Available Switches:

-InheritsFromParent	Inherited permissions from the parent object, which changes will be reset.
-Remove	Removes the appropriate user completely from the permissions.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Set the permission "Full" for the user group (32-bit hive)
SF-RegPerm32 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "%_GroupUsers%"

# Set the permission "Full" for the user group (32-bit hive)
SF-RegPerm32 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "%_GroupUsers%" "Read"

# Remove permission (32-bit hive)
SF-RegPerm32 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "%_GroupUsers%" -Remove

# Reset permissions (32-bit hive)
SF-RegPerm32 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" -InheritsFromParent
```

2.52 SF-RegPerm64

Overview:
Modifies the 64-bit registry permissions.

Description:
None

Syntax:
```
SF-RegPerm64 "RegHive" "SubKey" "User" -Switches
```

Available Switches:

-InheritsFromParent	Inherited permissions from the parent object, which changes will be reset.
-Remove	Removes the appropriate user completely from the permissions.

Examples:
```
# Set the permission "Full" for the user group (64-bit hive)
SF-RegPerm64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "%_GroupUsers%"

# Set the permission "Full" for the user group (64-bit hive)
SF-RegPerm64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "%_GroupUsers%" "Read"

# Remove permission (64-bit hive)
SF-RegPerm64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" "%_GroupUsers%" -Remove

# Reset permissions (64-bit hive)
SF-RegPerm64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Example" -InheritsFromParent
```

2.53 SF-RegRead32

Overview:
Reads a registry value from the 32-bit hive.

Description:
The read value is stored as a variable Scripting Framework, which can be used subsequently in the script.

Syntax:
```
SF-RegRead32 "RegHive" "SubKey" "ValueName" "VariabelName" -Switches
```

Available Switches:

-Machine	Saves the machine-related variable under HKLM\Software\Scripting Framework\Variables
-Script	Creates a temporary PowerShell Variable
-User	Saves the user-related variable under HKCU\Software\Scripting Framework\Variables

Examples:
```
# Example 1 - Read a registry value (32-bit) to Scripting Framework variable %RegRead_Example%
SF-RegRead32 "HKey_Local_Machine" "SOFTWARE\Example" "ValueName" "RegRead_Example"

# Example 2 - Create New Scripting Framework Variable <New_VariableName>
SF-RegRead32 "HKEY_LOCAL_MACHINE" "SOFTWARE\Microsoft\Windows\CurrentVersion" "CommonFilesDir" "New_SFVariable"
If ((SF-Variables "%New_SFVariable%") -eq "C:\Program Files (x86)\Common Files")
{
    SF-Log "Value is equal <C:\Program Files (x86)\Common Files>"
}
Else {
    SF-Log "Value is NOT equal <C:\Program Files (x86)\Common Files>"
}

# Example 3 - Create New PowerShell Variable $New_PSVariable
SF-RegRead32 "HKEY_LOCAL_MACHINE" "SOFTWARE\Microsoft\Windows\CurrentVersion" "CommonFilesDir" "New_PSVariable" -Script
SF-Log "PSVariable: $($New_PSVariable)"
```

2.54 SF-RegRead64

Overview:
Reads a registry value from the 64-bit hive.

Description:
The read value is stored as a variable Scripting Framework, which can be used subsequently in the script.

Syntax:
```
SF-RegRead64 "RegHive" "SubKey" "ValueName" "VariabelName" -Switches
```

Available Switches:

-Machine	Saves the machine-related variable under HKLM\Software\Scripting Framework\Variables
-Script	Creates a temporary PowerShell Variable
-User	Saves the user-related variable under HKCU\Software\Scripting Framework\Variables

Examples:
```
# Example 1 - Read a registry value (32-bit) to Scripting Framework variable %RegRead_Example%
SF-RegRead64 "HKey_Local_Machine" "SOFTWARE\Example" "ValueName" "RegRead_Example"

# Example 2 - Create New Scripting Framework Variable <New_VariableName>
SF-RegRead64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Microsoft\Windows\CurrentVersion" "CommonFilesDir" "New_SFVariable"
If ((SF-Variables "%New_SFVariable%") -eq "C:\Program Files (x86)\Common Files")
{
    SF-Log "Value is equal <C:\Program Files (x86)\Common Files>"
}
Else {
    SF-Log "Value is NOT equal <C:\Program Files (x86)\Common Files>"
}

# Example 3 - Create New PowerShell Variable $New_PSVariable
SF-RegRead64 "HKEY_LOCAL_MACHINE" "SOFTWARE\Microsoft\Windows\CurrentVersion" "CommonFilesDir" "New_PSVariable" -Script
SF-Log "PSVariable: $($New_PSVariable)"
```

2.55 SF-RegSearch32

Overview:
The function searches the registry (32-bit hive) for a value and can optionally check for the value or name. The search is performed recursively, and the function returns true or false as the value. If any is resembled the RegHive and SubKey passed to the function, check for the existence of the key takes place. The result is also in a multidimensional array (SearchList) saved, thereby further evaluations are possible.

Description:
None

Syntax:
```
SF-RegSearch32 "RegHive" "SubKey" "SearchText" "ValueName" -Switches
```

Available Switches:

-ExcludeKeys	By default, is searched in the key name after the text. This parameter specifies the key name may be excluded from the search.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Searches for "SearchString" under the Key "Software\Example"
If ((SF-RegSearch32 "HKEY_LOCAL_MACHINE" "Software\Example" "SearchString") -eq $True) {
	SF-Log "Value found"
}

# Searches for "SearchString" under the Key "Software\Example" the value name must be "DisplayName". The function returns true if the text was found
If ((SF-RegSearch32 "HKEY_LOCAL_MACHINE" "Software\Example" "SearchString" "DisplayName") -eq $True) {
	SF-Log "Value found"
}

# Search with wildcards
If ((SF-RegSearch32 "HKEY_LOCAL_MACHINE" "Software\Example" "*SearchString*") -eq $True) {
	SF-Log " Value found "
}

# Search and subsequently evaluating the array
SF-RegSearch32 "HKEY_LOCAL_MACHINE" "Software\Example" "SearchString"

# First Array Entry
If ($SearchList -ne $null)
{
```

```
    SF-Log $SearchList[0][0] # RootKey
    SF-Log $SearchList[0][1] # SubKey
    SF-Log $SearchList[0][2] # ValueName
    SF-Log $SearchList[0][3] # Value
}

# All Array Entries
ForEach($Search in $SearchList)
{
    SF-Log $Search
}
```

2.56 SF-RegSearch64

Overview:
The function searches the registry (64-bit hive) for a value and can optionally check for the value or name. The search is performed recursively, and the function returns true or false as the value. If any is resembled the RegHive and SubKey passed to the function, check for the existence of the key takes place. The result is also in a multidimensional array (SearchList) saved, thereby further evaluations are possible.

Description:
None

Syntax:
```
SF-RegSearch64 "RegHive" "SubKey" "SearchText" "ValueName" -Switches
```

Available Switches:

-ExcludeKeys	By default, is searched in the key name after the text. This parameter specifies the key name may be excluded from the search.

Examples:
```
# Searches for "SearchString" under the Key "Software\Example"
If ((SF-RegSearch64 "HKEY_LOCAL_MACHINE" "Software\Example" "SearchString") -eq $True) {
       SF-Log "Value found"
}
```

```powershell
# Searches for "SearchString" under the Key "Software\Example" the value name must be "DisplayName". The function returns
true if the text was found
If ((SF-RegSearch64 "HKEY_LOCAL_MACHINE" "Software\Example" "SearchString" "DisplayName") -eq $True) {
	SF-Log "Value found"
}


# Search with wildcards
If ((SF-RegSearch64 "HKEY_LOCAL_MACHINE" "Software\Example" "*SearchString*") -eq $True) {
	SF-Log "Gefunden"
}

# Search and subsequently evaluating the array
SF-RegSearch64 "HKEY_LOCAL_MACHINE" "Software\Example" "SearchString"

# First Array Entry
If ($SearchList -ne $null)
{
    SF-Log $SearchList[0][0] # RootKey
    SF-Log $SearchList[0][1] # SubKey
    SF-Log $SearchList[0][2] # ValueName
    SF-Log $SearchList[0][3] # Value
}

# All Array Entries
ForEach($Search in $SearchList)
{
	SF-Log $Search
}
```

2.57 SF-Rename

Overview:
To rename a folder or file.

Description:
None

Syntax:

```
SF-Rename "Source" "Target" -Switches
```

Available Switches:

-Force	If the TargetPath exists, it will be removed before renaming
-LocalUserProfiles	Renames a file recursively in all local user profiles

Examples:

```
# Rename folder to Example1
SF-Rename "%_SystemDrive%\Example" "%_SystemDrive%\Example1"

# Rename file to Example1.ini
SF-Rename "%_SystemDrive%\Example\Example.ini" "%_SystemDrive%\Example\Example1.ini"

# Renames the file Example.txt to Example2.txt in all local user profiles
SF-Rename "AppData\Roaming\Example.txt" "AppData\Roaming\Example2.txt" -LocalUserProfiles
```

2.58 SF-Replace

Overview:
Finds and replaces a value and writes the new value as Scripting Framework variable in the registry.

Description:
None

Syntax:
```
SF-Replace "Value" "Search" "Replace" "VariableName" -Switches
```

Available Switches:

-User	The variable is defined for the user HKCU\Software\Scripting Framework\Variables

Examples:
```
# Replace the word input with output in the variable Example
SF-Replace "InputText" "Input" "Output" "Example"

# Writes the value C:/Windows in the variable Example
SF-Replace "%_Windows%" "\" "/" "Example"
```

2.59 SF-Requirements

Overview:
Checks the system requirements.

Description:
The parameters are used to check whether the system fulfills the conditions. If a condition is not met, the script is terminated.

Syntax:
```
SF-Requirements -Switches
```

Available Switches:

-CPUSpeed	Checks the minimum CPU speed in MHz.
-Date "mm/dd/yyyy H:mm"	Checks if the defined date and time is reached.
-FreeDiskSpaceSystemDrive	Checks the minimum free disk space on the system drive (C:\) in MB.
-TotalPhysicalMemory	Checks the minimum physical memory in MB.
-x86	Checks if it is a 32-Bit operating system.
-x64	Checks if it is a 64-Bit operating system.

Examples:
```
# Example 1 - Checks only the free disk space of the system drive
SF-Requirements -FreeDiskSpaceSystemDrive "2000"

# Example 2 - Checks the free disk space of the system drive and the physical memory
SF-Requirements -FreeDiskSpaceSystemDrive "2000" -TotalPhysicalMemory "4000"

# Example 3 - Checks the free disk space of the system drive, the physical memory and OS architecture
SF-Requirements -FreeDiskSpaceSystemDrive "2000" -TotalPhysicalMemory "4000" -x64

# Example 4 - Checks the date
SF-Requirements -Date "06/22/2019 12:00"
```

2.60 SF-Run

Overview:
Starts a file you want and evaluates the return code from automatically.

Description:
By default runs the Run command on an error, unless one of the following exit codes is returned (0, 1605, 1641 and 3010). By a 1641 or 3010, is automatically to the SF-MSIInstall command a reboot registered (eg InstallShield installations). An error can be suppressed with parameter -NoErrors. Thus, it is possible to evaluate the return code on the PowerShell variable $ExitCode even see example. It is also possible via the ExcludedExitCodes parameters to ignore other codes, they must be comma indicated separately.

Syntax:
```
SF-Run "Path" "Arguments" "WorkingDirectory" -Switches
```

Available Switches:

-ExcludedExitCodes	With this parameter you can specify additional return codes, which are not regarded as errors.
-LoggedOnUsers	This parameter allows the process to automatically start on all logged-on users with their credentials, even if the package is installed under the SYSTEM account. This allows a process to be started and displayed to the user even if the installation of the package runs under the SYSTEM account. If no user is logged-on, the run command is skipped.
-NoErrors	Error messages are ignored and the script continues.
-Show	The window is displayed and not as hidden started.
-SkipIfNotExist	If the file does not exist, the script continues without error.
-Timeout x	In conjunction with the Switch Wait, a timeout (maximum wait time) can be specified in seconds. If the process runs longer than the defined timeout, exit code 6010 is returned.
-Wait	Wait until the process has been completed.
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Run Notepad without wait and show the interface
SF-Run "%_Windows%\notepad.exe" "" -Show

# Run Notepad on a x86 OS
SF-Run "%_Windows%\notepad.exe" "" –Show –x86
```

```
# Run Notepad and waiting until it is closed, and check the return code manually
SF-Run "%_Windows%\notepad.exe" "" -Show -Wait -NoErrors
If (($ExitCode -eq 3) -or ($ExitCode -eq 4)) {
	SF-Exit "Error detected"
}

# Run Notepad and excluding other return codes
SF-Run "%_Windows%\notepad.exe" "" -Show -Wait -ExcludedExitCodes "256,1024"


# Start notepad and wait 10 seconds maximum
SF-Run "%_Windows%\notepad.exe" "" -Show -Wait -Timeout 10
```

2.61 SF-Service

Overview:
This command controls a Windows Service.

Description:
It is possible to start this, stop, restart, or even delete them. In addition, the startup type to automatic, delayed, Disabled or Manual.

Syntax:
```
SF-Service "Servicename" -Switches
```

Available Switches:

-Start	Start a service
-Stop	Stops a service
-Restart	Restart a service
-Automatic	Set the startup type to Automatic
-Delayed	Set the startup type to Automatic (Delayed Start)
-Disabled	Set the startup type to Disabled
-Manual	Sets the startup type to Manual
-Delete	Deletes the service during the next reboot

Examples:
```
# Set the startup type to automatic
SF-Service "spooler" –Automatic -Start

# Stop the service
SF-Service "spooler" –Stop

# Restart the service
SF-Service "spooler" -Restart

# Set the startup type to Automatic and restart the service
SF-Service "spooler" –Automatic -Restart
```

2.62 SF-ServiceInstall

Overview:
This command installs a Windows Service.

Description:
None

Syntax:
```
SF-ServiceInstall "Binary" "ServiceName" "DisplayName" "Description" "Username" "Password" -Switches
```

Available Switches:

Username	The username parameter is optional
Password	The Password is a optional parameter
-Automatic	Set the startup type to Automatic
-Delayed	Set the startup type to Automatic (Delayed Start)
-Disabled	Set the startup type to Disabled
-Restart	Restart a service
-Manual	Set the startup type to Manual

Examples:
```
# Creates a service with the startup type Automatic (runs with the system account)
SF-ServiceInstall "%_SystemDrive%\Example\MyService.exe" "ServiceExample" "My example service" "This is a sample service"
"" "" -Automatic

# Creates a service with the startup type Automatic (runs with the user ExampleUser1)
SF-ServiceInstall "%_SystemDrive%\Example\MyService.exe" "ServiceExample" "My example service" "This is a sample service"
"ExampleUser1" "Password" -Automatic
```

2.63 SF-ServiceStatus

Overview:
This command checks the status of a Windows Service.

Description:
It is possible to check whether the service exists, or whether the status or StartType corresponds to a certain value. The command returns true or false. The command can also return the value StartType or Status as text.

Syntax:
```
SF-ServiceStatus "ServiceName" -Switches
```

Available Switches:

-StartType	Specifies which StartType of the service is queried. The acceptable values for this parameter are: Automatic, AutomaticDelayedStart, Disabled, Manual
-Status	Specifies which Status of the service is queried. The acceptable values for this parameter are: Running, Stopped, Paused
-ReturnStartType	Specifies that the StartType is returned as text by the specified service.
-ReturnStatus	Specifies that the Status is returned as text by the specified service.
-NoErrors	If the service does not exist, the error is ignored.

Examples:
```
# Checks if the service is exists
If ((SF-ServiceStatus "spooler") -eq $true) {
    SF-Log "The Service exists"
}
Else {
    SF-Log "The service does not exist"
}

# Checks the service StartType
If ((SF-ServiceStatus "spooler" -StartType Automatic) -eq $true) {
    SF-Log "The Service exists and the StartType is Automatic"
}
Else {
    SF-Log "The service does not exist or the StartType is not Automatic."
}
```

```
# Checks the service StartType and Status
If ((SF-ServiceStatus "spooler" -StartType Automatic -Status Running) -eq $true) {
    SF-Log "The Service exists, the StartType is Automatic and the Status is Running"
}
Else {
    SF-Log "The service does not exist, the StartType is not automatic or the status is not running."
}

# Returns the status as value
$Result = SF-ServiceStatus "spooler" -ReturnStatus
If ($Result -eq "Running") {
    SF-Log "Service is Running"
}
```

2.64 SF-Set

Overview:
Sets a Scripting Framework variable, which is stored in the registry.

Description:
The variable can be approximated with %name% in the script.

Syntax:
```
SF-Set "Variabelname" "Value" -Switches
```

Available Switches:

-Delete	Deletes the variable
-Environment	Sets a Variable Environment (System or User). To set a User Environment Variable The User parameter must be given in addition.
-Script	Sets a temporary variable, which is valid only within the script
-User	The variable is defined for the user HKCU\Software\Scripting Framework\Variables
-UseVarPrefix	When creating the variable, a prefix is automatically used for the variable name (a_PkgIdentifier-NAME). The value of the variable can be addressed in the script with %_VarPrefix-NAME%.

Examples:
```
# Sets the machine variable Example with a value of C:\Windows
SF-Set "Example" "%_Windows%"

# Set the system environment variable Example with the value C:\Windows
SF-Set "Example" "%_Windows%" -Environment

# Sets the user variable Example with the value %AppData%
SF-Set "Example" "%_ApplicationData%" -User

# Removes the machine variable
SF-Set "Example" -Delete

# Removes the system environment variable
SF-Set "Example" -Environment -Delete

# Removes the user variable
SF-Set "Example" -Delete –User
```

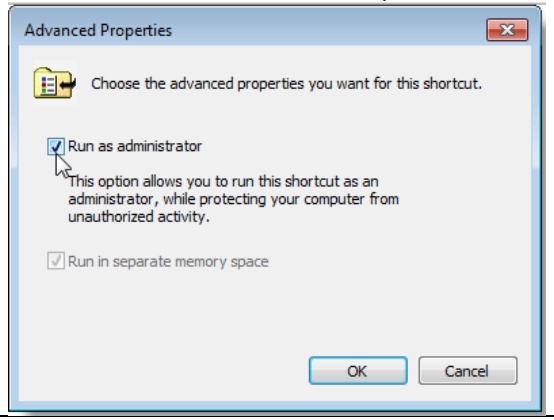
2.65 SF-Shortcut

Overview:
Creates a shortcut with the corresponding information.

Description:
In addition to the target should be a parameter is necessary, this will be separated with a comma (see example).

Syntax:
```
SF-Shortcut "Shortcut" "Target" "Start in" "Icon" "Comment" -Switches
```

Available Switches:

-Elevated	Sets for the created shortcut the option „Run as administrator“. 
-Hotkey	Define a hotkey (key combination) for the shortcut. For example "CTRL+SHIFT+F1"
-Maximized	Defines the window style as Maximized
-Minimized	Defines the window style as Minimized

Examples:
```
# Create a shortcut for notepad
SF-Shortcut "%_CommonPrograms%\Notepad.lnk" "%_Windows%\notepad.exe" "%_Windows%" "%_Windows%\notepad.exe,0" "Notepad
Example"

# Create a shortcut with parameters
SF-Shortcut "%_CommonPrograms%\Notepad with Parameter.lnk" "%_Windows%\notepad.exe,%_Windows%\WindowsUpdate.log"
"%_Windows%" "%_Windows%\notepad.exe,0" "Notepad Example" -Maximized -Hotkey "CTRL+SHIFT+F1"
```

2.66 SF-ShowBalloonTip

Overview:
Displays a balloon tip in the lower right corner of the screen. The command must be used in the current user context (no BalloonTip is displayed if the command is executed in another user context).

Description:
Displays a balloon tip in the lower right corner of the screen. Title, text and severity can be customized.

Syntax:
```
SF-ShowBalloonTip -Switches
```

Verfügbare Switches:

-Severity	Defines the severity of the message. Valid values are: Error, Info, Warning and None

Examples:
```
# Displays a balloon tip
SF-ShowBalloonTip "Scripting Framework" "This is a example balloon tip" -Severity Info
```

2.67 SF-ShowCustomNotification

Overview:
Shows to the user a custom message (this works also under the System account).

Description:
The function displays a custom Scripting Framework Notification. The text elements and buttons can be configured via the parameters. The arrangement of the buttons is from right to left. The return value is the clicked button, the possible return values are Button1, Button2 or Button3.

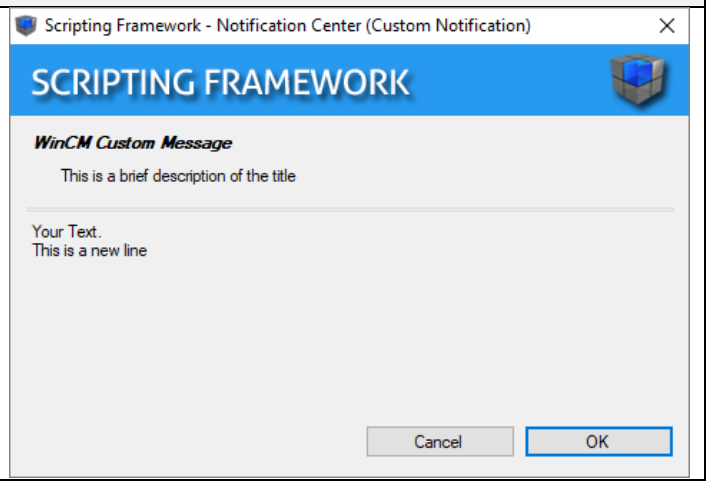
Syntax:
```
SF-ShowCustomNotification "Titel" "TitelDescription" "Text" -Switches
```

Available Switches:

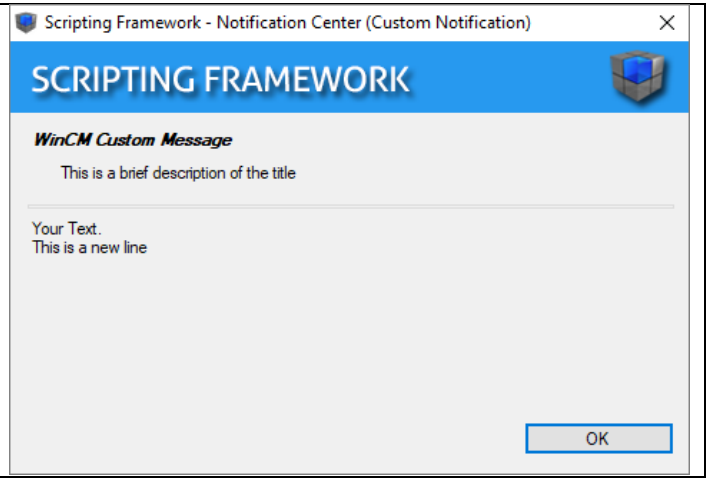
-Button1 x	Displays Button1 with the defined text.
-Button2 x	Displays Button2 with the defined text.
-Button3 x	Displays Button3 with the defined text.
-Wait	Wait until the user clicks a button. Attention: Without the Wait parameter the result can not be evaluated, in this case it is always OK!

Examples:
```
# Displays a notification with two buttons, then the result is
evaluated.
$Result = SF-ShowCustomNotification "WinCM Custom Message" "This is a
brief description of the title" "Your Text. `nThis is a new line" -
Button1 "OK" -Button2 "Cancel" -Wait
```

Notification:



```powershell
# Displays a notification with one button without waiting for the result
SF-ShowCustomNotification "WinCM Custom Message" "This is a brief
description of the title" "Your Text. `nThis is a new line" -Button1
"OK"
```



2.68 SF-ShowInstallNotification

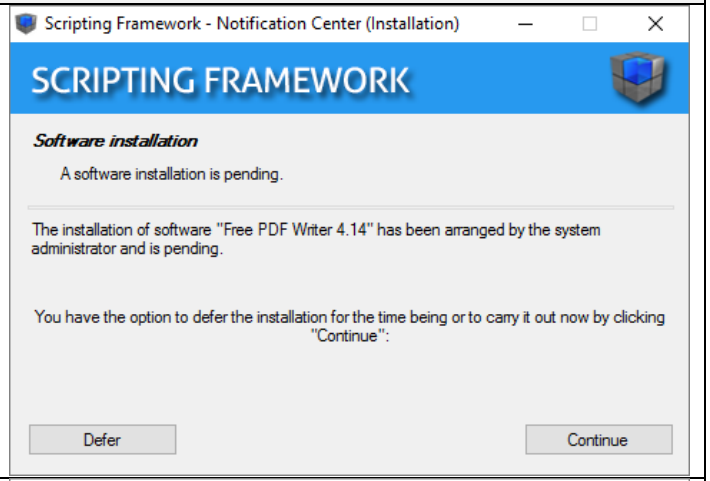
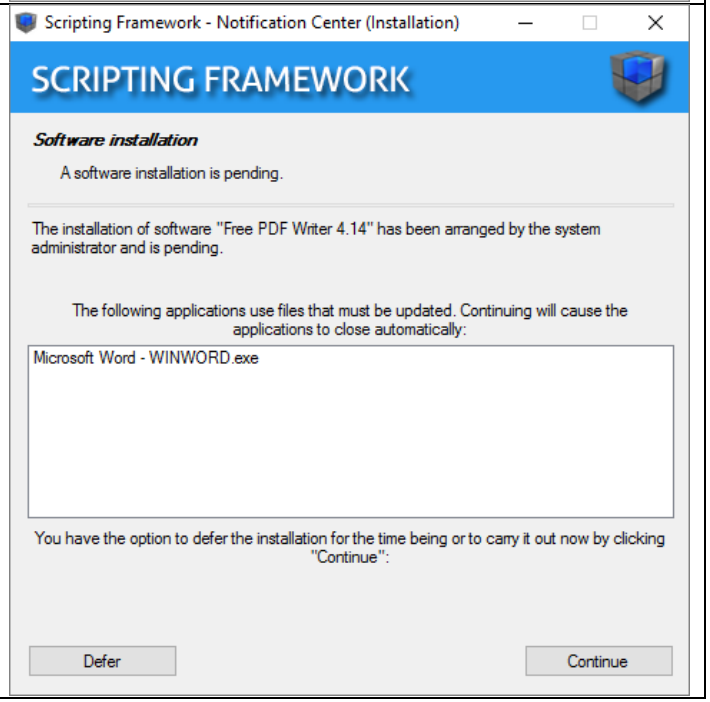
Overview:
Shows the user a notification which heralds the installation of the package (this works also under the System account). See also Scripting Framework Notification Center.

Description:
The registered users will be informed via notification that the installation of a software package is pending. The user can decide whether the installation run at a later time or immediately. It is optionally possible to determine the number of shifts and/or the key date. It is also possible to inform the user about the closed tasks. If no user is logged on the machine, the installation will continue automatically without a message. On displacement of the installation, Scripting Framework returns the return code 70100.

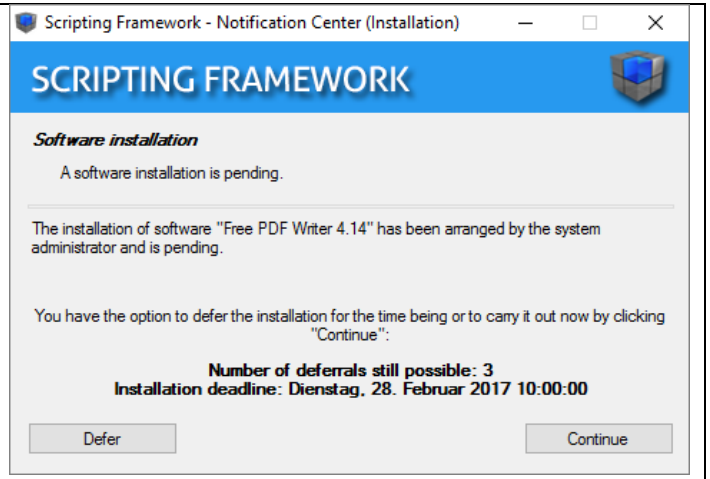
Syntax:
```
SF-ShowInstallNotification -Switches
```

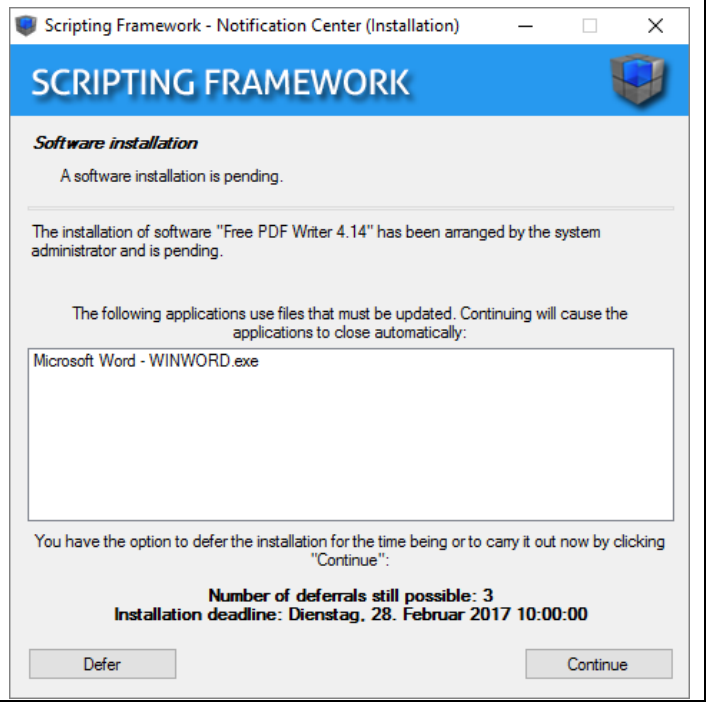
Available Switches:

-Description	Optional can be defined instead of the Description (from the language.xml) a separate text. With ## NAME ## it is possible to output the package name. See examples.
-DefaultActionDefer x	Timeout in seconds, until the installation is shifted automatically, unless the user does not select anything. Without this parameter, no timeout is set (value 0). The timeout countdown is visible on the corresponding button.
-DefaultActionContinue x	Timeout in seconds, until the installation is performed automatically, as long as the user does not select anything. Without this parameter, no timeout is set (value 0). The timeout countdown is visible on the corresponding button.
-DeferCounter x	With this parameter you can define the desired number to restrict the number of possible shifts. A combination with the parameter DeferDeadline is possible.
-DeferDeadline "mm/dd/yyyy H:mm"	The date and time which no further displacement of the installation is possible.
-DeferDisabled	Hides the „Defer” button on the GUI.
-Reappear x	When the installation is shifted, the Notification Center is minimized and not closed. After x seconds it is displayed again.
-Taskkill x	Defines the tasks to be closed for installation. With the analog command SF-Taskkill it is possible to define multiple processes or directories with the separator “|”.The notification center checks whether one of the specified processes is active. If no active process are found, a notification without running applications is displayed.
-ShowOnlyIfTaskkill	The Notification is displayed only if active processes exist, which have been defined with the parameter Taskkill. Otherwise, the installation of the package will be continued without informing the user.

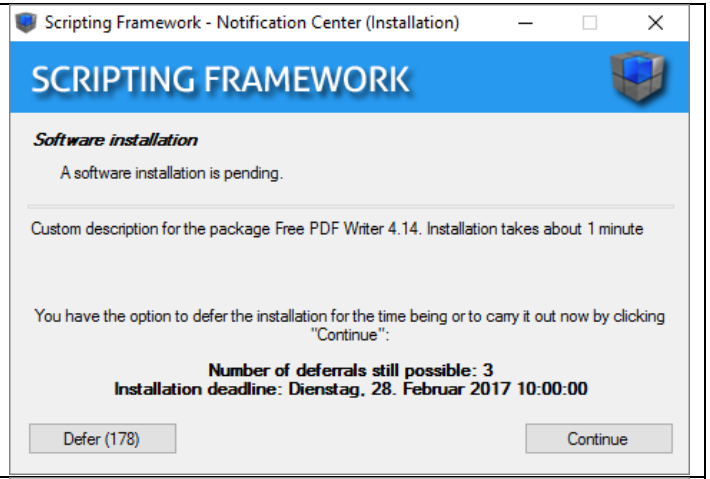
Examples:	Notification:
`# Default notification without parameters` `SF-ShowInstallNotification`	
`# Default notification with killing tasks (only the active tasks are displayed to the user). In this example, Microsoft Word was found` `SF-ShowInstallNotification -Taskkill "winword.exe|excel.exe"`	

```
# Notification with a taskkill (all processes in a folder and two
individual tasks), maximum defer 3 times, and installation deadline is
2.01.2016. The first screenshot shows the notification, if there is no
active process has been found. On the second screenshot was Microsoft
Word active.
SF-ShowInstallNotification -Taskkill "%_ProgramFiles32%\Internet
Explorer|winword.exe|excel.exe" -DeferCounter 3 -DeferDeadline
"02/28/2017 10:00"
```

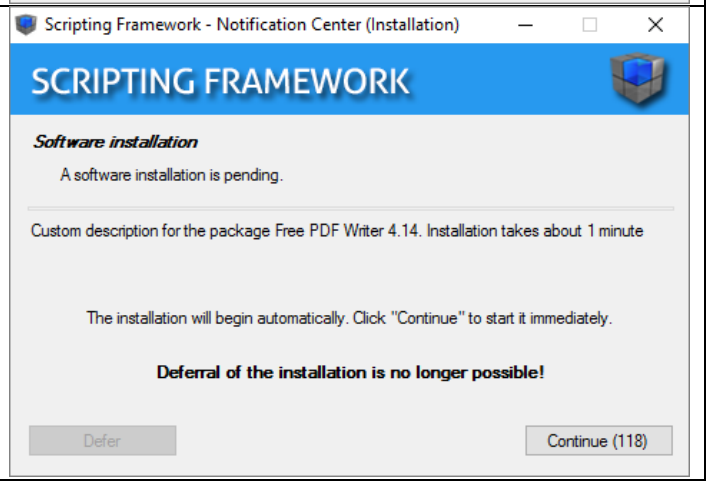




```
# Notification with a custom description. With ##NAME## you can show the package installation name.
SF-ShowInstallNotification -Description "Custom description for the package ##NAME##. Installation takes about 1 minute" -DeferCounter 3 -DeferDeadline "02/28/2017 10:00" -DefaultActionDefer 180
```




```
# Notification when deadline is reached (no task to kill)
```



2.69 SF-ShowProgressBar

Overview:
Shows a ProgressBar to the user during install or uninstall (this also works under the system account).

Description:
The command can be used to show the user a progress bar to indicate to the user that an install or uninstall is still running in the background. This can be particularly useful for long installation times. After completing the package, the ProgressBar will be closed automatically.

Syntax:
```
SF-ShowProgressBar -Switches
```

Available Switches:

-ExpectedTime x	Specifies the expected time in seconds, how long the process will take approximately. This affects how fast the progress bar is to 100%. Once 100% has been reached, the text display is automatically adjusted to indicate that the process will soon be completed. If the parameter is not specified, a repeating loading bar is displayed.
-IfSYSTEM	The command is only executed if the package is run under the SYSTEM account context, otherwise the command is skipped.
-Stop	Exits the ProgressBar before the package completes.

Examples:
```
# Displays a ProgressBar to the user
SF-ShowProgressBar

# Shows the user a ProgressBar with the progress bar reaching 100 after 60 seconds
SF-ShowProgressBar -ExpectedTime 60

# Stop displaying the ProgressBar early
SF-ShowProgressBar -Stop
```

2.70 SF-ShowSCCM

Overview:
Shows the user the Microsoft System Center Client (this also works under the system account).

Description:
The System Center Client (SFClient.exe) will start on all logged on users to show the installation status.

Syntax:
```
SF-ShowSCCM
```

Available Switches:

None	

Examples:
```
# Shows the System Center client to the user
SF-ShowSCCM
```

2.71 SF-Supersedence

Overview:
The function is used to replace an already installed Scripting Framework package within a new package without uninstalling it. By using the function, all entries relevant to the Scripting Framework (variables, etc.) within the package that is to be replaced are cleared. If the installation exists, “true” is given as the return value; otherwise, “false” is given. The return value makes appropriate queries and actions possible.

Description:
As a parameter, the function is given the identifier of the package that is to be replaced. Wildcards are also supported for this purpose (see examples). The PkgIdentifier is used to check whether the corresponding Scripting Framework package is installed on the client. If the package that is to be replaced is found, all entries relevant to the Scripting Framework are cleared, and “true” is given as the return value. The return value makes it possible to react within the script with a simple query and trigger actions accordingly. For example, if a package that is to be replaced is detected, it is possible to install only a single patch instead of a complete software (see examples).

Syntax:
```
SF-Supersedence "Identifier" -Switches
```

Available Switches:

-Cleanup	With this switch, the corresponding application key under „HKLM\SOFTWARE\ *WOW6432Node*\ScriptingFramework\Inventory\Applications\PkgIdentifier“ is completely removed. Without the use of the parameter, the InstallState value is set to “uninstalled” and the UninstallDate is set. This behavior corresponds to that of uninstalling a package through Uninstall.exe.

Examples:
```
# Sample 1 – Clears the entries of the package and sets the InstallState in the application inventory to “uninstalled”
SF-Supersedence "Adobe_Acrobat_Reader_DC_1501020060_MUI_UNV_01"

# Sample 2- Clears the entries of all version 15 packages (wildcard *) and removes the package from the application
inventory
SF-Supersedence "Adobe_Acrobat_Reader_DC_15*" -Cleanup
```

```
# Sample 3 - Replacement of Adobe Reader 15.010.20060 with version 15.020.20039 with the use of a query
If ((SF-Supersedence "Adobe_Acrobat_Reader_DC_1501020060_MUI_UNV_01")) {
    # Only Patch Installation
    SF-MSIPatchInstall "%_PkgSource%\Setup\Patches\AcroRdrDCUpd1502020039_MUI.msp" "ALLUSERS='1' /qn /norestart"
}
Else {
    # Full Installation with Patch
    SF-MSIInstall "%_PkgSource%\Setup\AcroRead.msi" "ALLUSERS='1' EULA_ACCEPT='YES' /qn /norestart"
    SF-MSIPatchInstall "%_PkgSource%\Setup\Patches\AcroRdrDCUpd1502020039_MUI.msp" "ALLUSERS='1' /qn /norestart"
}
```

2.72 SF-SymbolicLink

Overview:
Creates a Symbolic link to a file or directory.

Description:
Creates a Symbolic link to a file or directory as an alternative to mklink.exe.

Syntax:
```
SF-SymbolicLink "Path" "Target"
```

Available Switches:

None	

Examples:
```
# Creates a symbolic link to Example folder that resides on C:\Program Files (x86)\Example.
SF-SymbolicLink "%_SystemDrive%\Example" "%_ProgramFiles32%\Example"

# Creates a symbolic link to Document.txt file under the directory C:\Program Files (x86) called SomeDocument.
SF-SymbolicLink "%_SystemDrive%\Example\Document.txt" "%_ProgramFiles32%\SomeDocument"

# Creates a symbolic link to Document.txt file under the current directory called SomeDocument.
SF-SymbolicLink "%_SystemDrive%\Example\Document.txt" "SomeDocument"
```

2.73 SF-SystemReboot

Overview:
The command detects a restart within the script, which for a successful end of the return code 3010 (reboot) is returned. The restart itself is not initiated from Scripting Framework! See Details.

Description:
None

Syntax:
```
SF-SystemReboot -Switches
```

Available Switches:

-DuringInstallation	The script is terminated without further execute code with the return code 1641. In conjunction with SCCM, a reboot is triggered before further installation processes. If the criteria of the detection methods are not fulfilled, the installation start again. In this caseyou need a flag in the script so that the reboot is performed only once.
-Immediate	At the end of the package installation a return code 1641 is returned.

Examples:
```
# Register a reboot
SF-SystemReboot

# Register a reboot immediate
SF-SystemReboot -Immediate

# Reboot during installation
If ((SF-Variables "%Reboot_Flag%") -ne "1") {
        SF-Set "Reboot_Flag" "1"
        SF-SystemReboot -DuringInstallation
}
```

2.74 SF-Taskkill

Overview:
With the Taskkill function can be a process terminated.

Description:
Instead of a single process can also a directory be specified to show all available processes and handles to close automatically. Several processes or folders are with the separation character are possible.

Syntax:
```
SF-Taskkill "Process or Folder"
```

Available Switches:

-Notification	If the application is running, the user see a message over the Scripting Framework Notification Center before the installation continues.
-Test	When using the switch, the function returns true or false, active processes are not closed.

Examples:
```
# Close Excel without a notification
SF-Taskkill "EXCEL.EXE"

# Close Excel with a notification
SF-Taskkill "EXCEL.EXE" -Notification

# Close Excel and Winword with a notification
SF-Taskkill "EXCEL.EXE|WINWORD.EXE" -Notification

# Close all processes in the Office14 and Reader 11.0 folder
SF-Taskkill "%_ProgramFiles32%\Microsoft Office\Office14|%_ProgramFiles32%\Adobe\Reader 11.0" -Notification

# Close specified process
SF-Taskkill "%_ProgramFiles64%\Java\jre1.8.0_77\bin\javaw.exe"

# Close Winword and all processes in Reader 11.0 folder
SF-Taskkill "WINWORD.EXE|%_ProgramFiles32%\Adobe\Reader 11.0" -Notification
```

2.75 SF-TestMSUpdate

Overview:
Check whether an Microsoft Update is installed on your system.

Description:
The function returns the value true or false.

Syntax:
```
SF-TestMSUpdate "Id"
```

Available Switches:

None	

Examples:
```
# Check if Microsoft Update installed
If ((SF-TestMSUpdate "KB2719306") -eq $True) {
    SF-Log "The update is installed"
}
```

2.76 SF-Textfile

Overview:
With the text file command text files can be manipulated/edited.

Description:
This includes the search and replace of a text that the replacement of an entire line or add text at the beginning or at the end of the file. It can also be defined that the text is added only if it does not already exist.

Syntax:
```
SF-Textfile "Path" "Text" "ReplaceWith" -Switches
```

Available Switches:

-Add	Adds the text you want at the end of the file.
-AtTop	Additional parameters for the Add. In order for the text at the beginning and not at the end of the file will be added.
-DeleteLine	Deletes the line in which the specified text is found.
-IfMissing	Add the text only if it does not already exist.
-Replace	Search and replace text.
-ReplaceLine	Replaces the entire line with a text if the word was found.
-TextFromFile	This parameter can be a text file instead of a text. In this file you can also use variables (e.g. %_PROGRAM FILES32%) is used. The source file will be read then line for line and written according to the desired file. All other parameters are also valid for this function!
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:
```
# Adds "Example Line" at the end of the file
SF-Textfile "%_SystemDrive%\Example\Example.txt" "Example Line" –Add

# Adds "Example Line" to the end of the file if it doesn't exist
SF-Textfile "%_SystemDrive%\Example\Example.txt" "Example Line" –Add –IfMissing

# Adds "Example Line" at the beginning of the file
SF-Textfile "%_SystemDrive%\Example\Example.txt" "Example Line" –Add –AtTop

# Adds the entire contents of "Source.text" file to Example.txt
SF-Textfile "%_SystemDrive%\Example\Example.txt" "%_SystemDrive%\Source.txt" –Add –IfMissing -TextFromFile
```

```powershell
# Search and replace the word Example with Example_2
SF-Textfile "%_SystemDrive%\Example\Example.txt" "Example" "Example_2" -Replace

# Replaces the complete line in which the text is found
SF-Textfile "%_SystemDrive%\Example\Example.txt" "Example" "Example_2" -ReplaceLine

# Clears the lines in which the text is found
SF-Textfile "%_SystemDrive%\Example\Example.txt" "Example" -DeleteLine
```

2.77 SF-UnRegisterFile32

Overview:
Unregisters a file.

Description:
Unregisters a 32-bit file.

Syntax:

```
SF-UnRegisterFile32 "Path"
```

Available Switches:

None	

Examples:

```
# Unregister file (32-Bit)
SF-UnRegisterFile32 "%_WindowsSystem32%\Example.dll"
```

2.78 SF-UnRegisterFile64

Overview:
Unregisters a file.

Description:
Unregisters a 64-bit file.

Syntax:
```
SF-UnRegisterFile64 "Datei"
```

Available Switches:

None	

Examples:
```
# Unregister file (64-Bit)
SF-UnRegisterFile64 "%_WindowsSystem64%\Example.dll"
```

2.79 SF-Variables

Overview:
Resolves the value of a Scripting Framework variable and returns it.

Description:
It is thus possible for example to the content of the variable must be checked, or it is possible the value of a variable in a Scripting Framework PowerShell variable to write.

Syntax:

```
SF-Variables "Variabel"
```

Available Switches:

None	

Examples:

```
# Check variable content
If ((SF-Variables "%_ProgramFiles32%") -eq "C:\Program Files (x86)") {
    SF-Log "String found"
}

# Generate a PowerShell variable
$Value = (SF-Variables "%_ProgramFiles32%")
```

2.80 SF-Wait

Overview:
With the wait command it is possible to achieve a desired time (in seconds) to wait. Can also be used for waiting for a starting or stopping process. With the specification of a file or a folder path you can wait so long until it exists.

Description:
None

Syntax:
```
SF-Wait "Arguments" -Switches
```

Available Switches:

-NoErrors	This parameter is relevant in connection with the timeout parameter. When the timeout is reached, the package is terminated with an error by default. With the -NoErrors parameter, the package continues with a warning.
-Timeout x	Defines the maximum wait time in seconds when specifying a process or a path to a file.
-TillDelete	If a file folder path specified as a parameter, it can be defined that will be waiting until the file or folder no longer exists.
-TillKill	If a process is specified as a parameter, it can be defined that will be waiting until the process is no longer running with this parameter.

Examples:
```
# Wait 5 seconds
SF-Wait "5"

# Wait until the file exists and then continue
SF-Wait "%_ProgramFiles32%\Example.txt"

# Wait until the file is deleted and then continue
SF-Wait "%_ProgramFiles32%\Example.txt " -TillDelete

# Waits for the Notepad process to start with a maximum wait time of 60 seconds
SF-Wait "Notepad.exe" -Timeout 60

# Wait until the process Notepad is closed and then continue
SF-Wait "Notepad.exe" -TillKill
```

2.81 SF-WUSAInstall

Overview:
Installs a Microsoft patch (.msu) and checks the appropriate return code (0, 1605, 1641, 3010, 2359302, -2145124329).

Description:
None

Syntax:

```
SF-WUSAInstall "Path" -Switches
```

Available Switches:

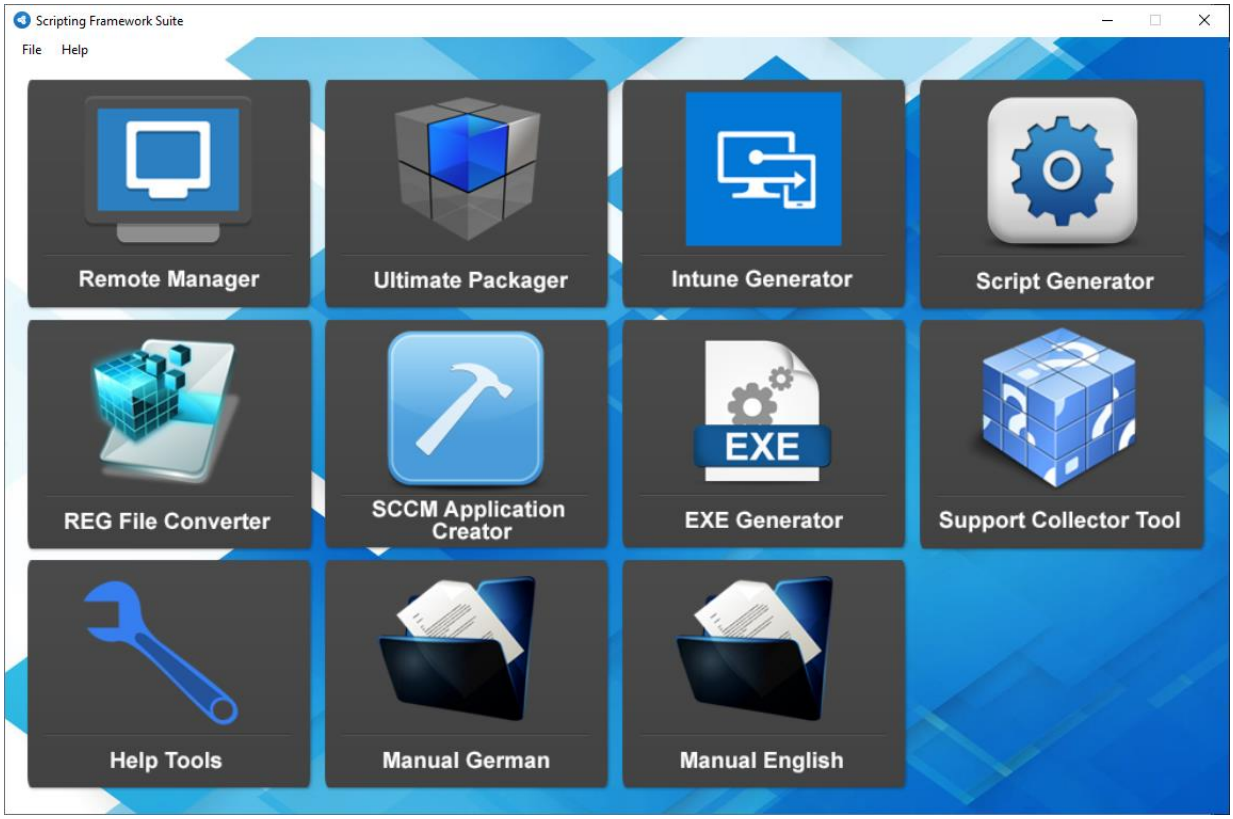
-Log	The MSI log is in the folder "C:\Windows\Scripting Framework\Logs\Software".
-x86	Run the command only on a 32-bit OS.
-x64	Run the command only on a 64-bit OS.

Examples:

```
# Install a MS hotfix
SF-WUSAInstall "%_PkgSource%\Setup\Windows6.1-KB2756822-x64.msu"

# Installs all hotfixes from the specified folder
SF-WUSAInstall "%_PkgSource%\Setup"
```

3 Available Suite Tools

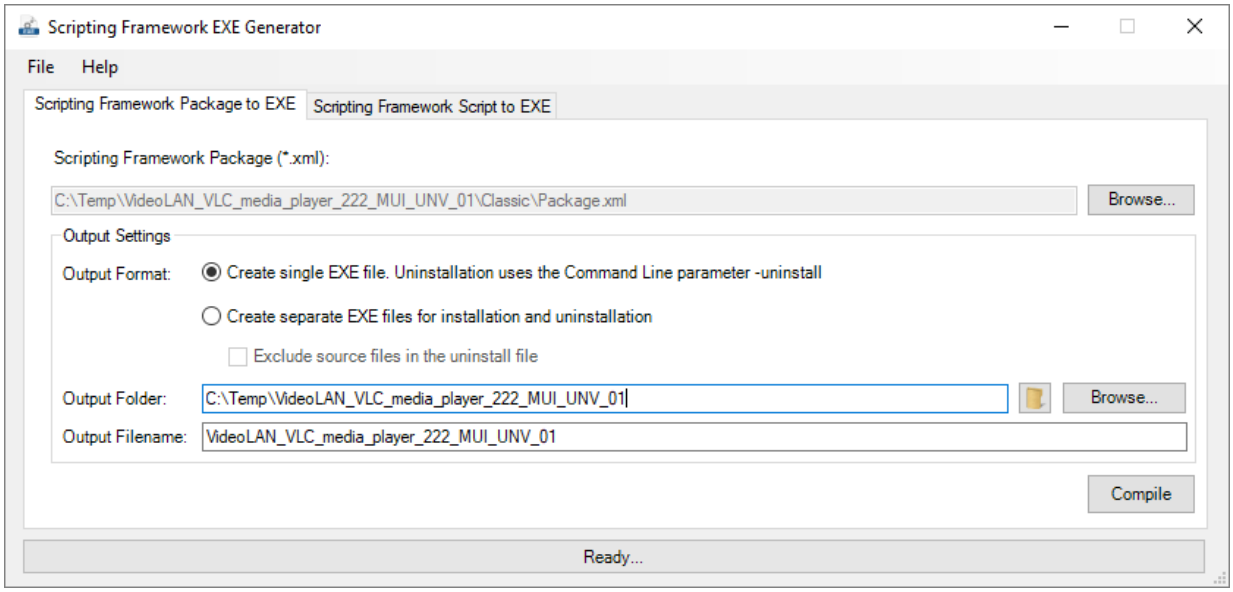
The Scripting Framework Suite includes all tools which are available as well as the Script Engine and which support you optimally. The suite includes for example the Ultimate Packager and the Script Generator, which allows you to quickly and easily create software packages based on the Scripting Framework functions.



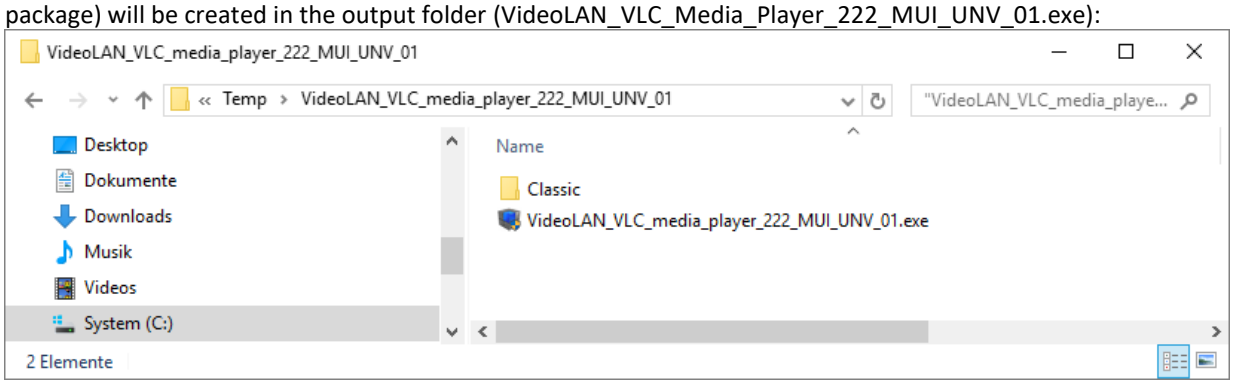
3.1 EXE Generator

From a Scripting Framework package or from a single script file, it's possible to create an executable file (.exe) with the „EXE Generator" tool (included in the Scripting Framework Suite). On the client, on which the executables are generated, the RARLAB WinRAR software with a valid license is needed. On the target client is WinRAR not needed, you need there only the Scripting Framework Engine.

3.1.1 Example Package To EXE

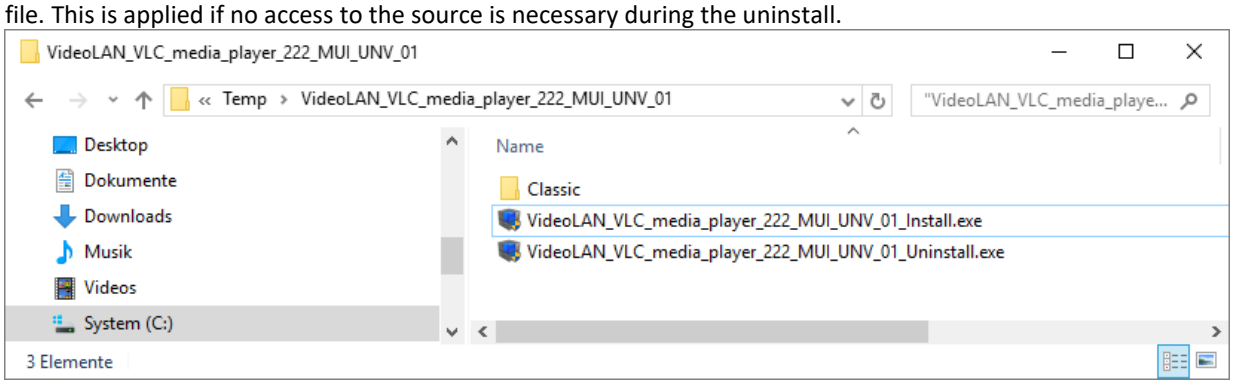


With the selected option on the screenshot „Create a single EXE file..." a single EXE file (from the selected package) will be created in the output folder (VideoLAN_VLC_Media_Player_222_MUI_UNV_01.exe):

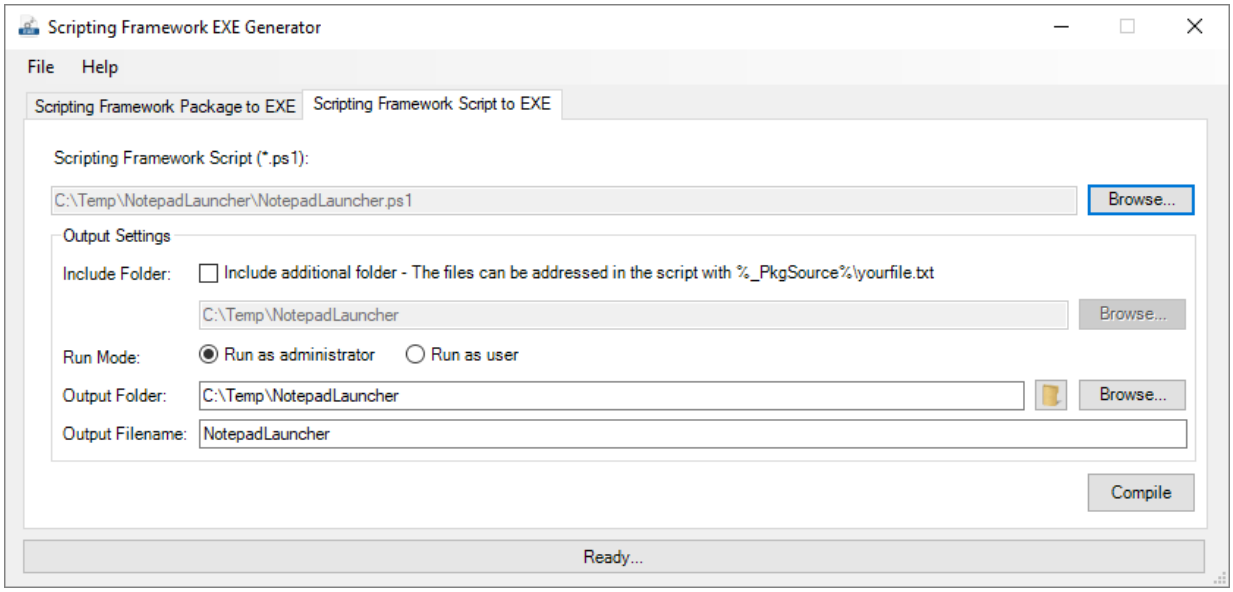


By calling the EXE file, the installation will start in silent mode, a parameter is not necessary. For the uninstallation is the command line parameter -uninstall available:
VideoLAN_VLC_Media_Player_222_MUI_UNV_01.exe -uninstall

The second available option "Create separate EXE files for installation and uninstallation" creates two separate files for installing and uninstalling. There is the possibility to exclude the Setup Source from the uninstalling EXE file. This is applied if no access to the source is necessary during the uninstall.

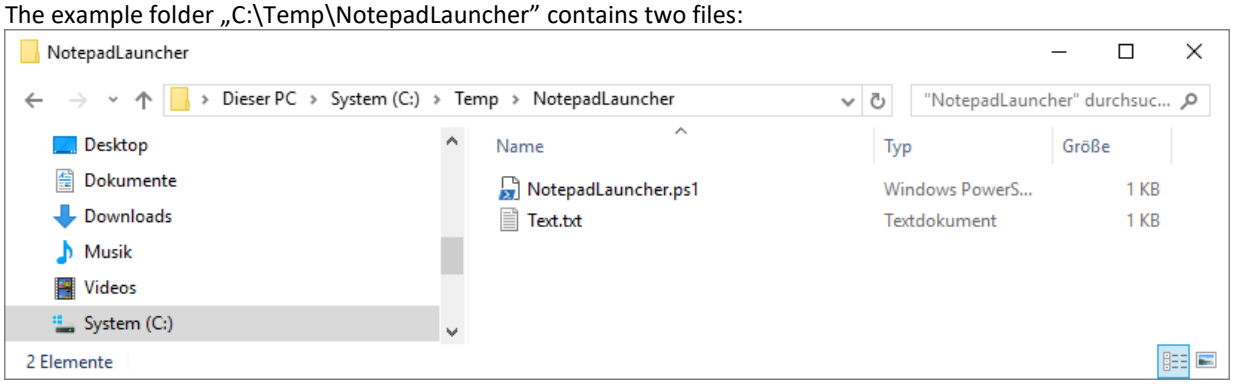


3.1.2 Example Script To EXE (Scripting Mode)

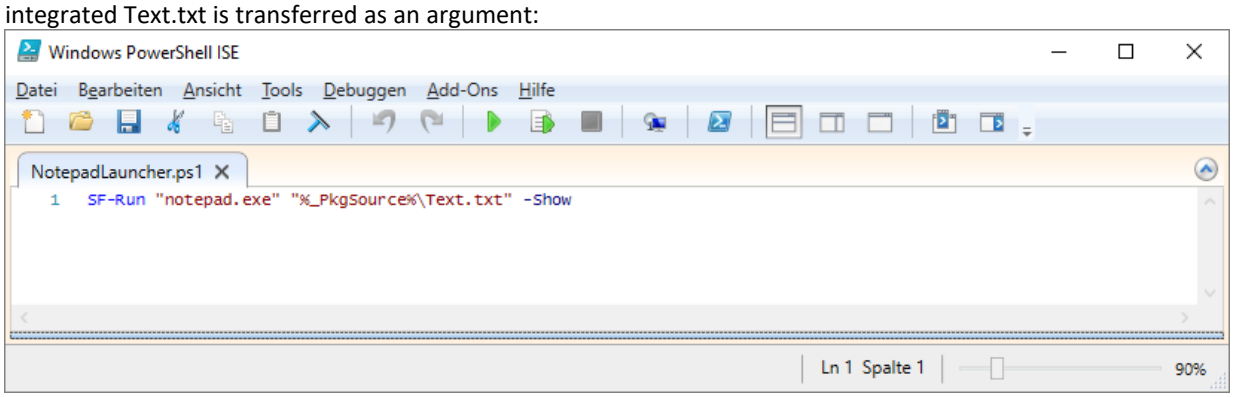


Under the tab „ Scripting Framework Scipt to EXE " a single script file (* .ps1) is compiled into an EXE, as opposed to an entire package. In addition to the script file, more files can be integrated and accessed directly from the script.

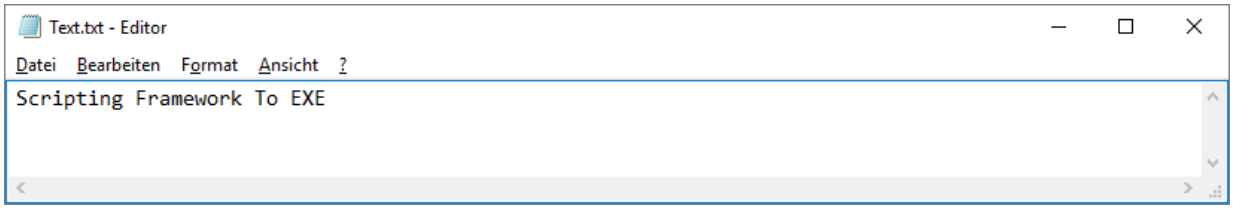
The example folder „C:\Temp\NotepadLauncher" contains two files:



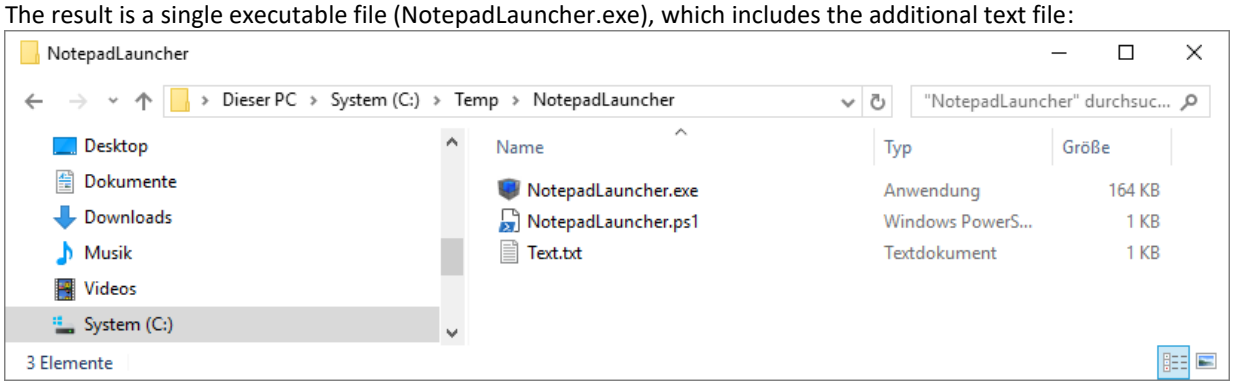
The script file „NotepadLauncher.ps1" contains the corresponding code in order to start Notepad, and the integrated Text.txt is transferred as an argument:



Content of Text.txt:



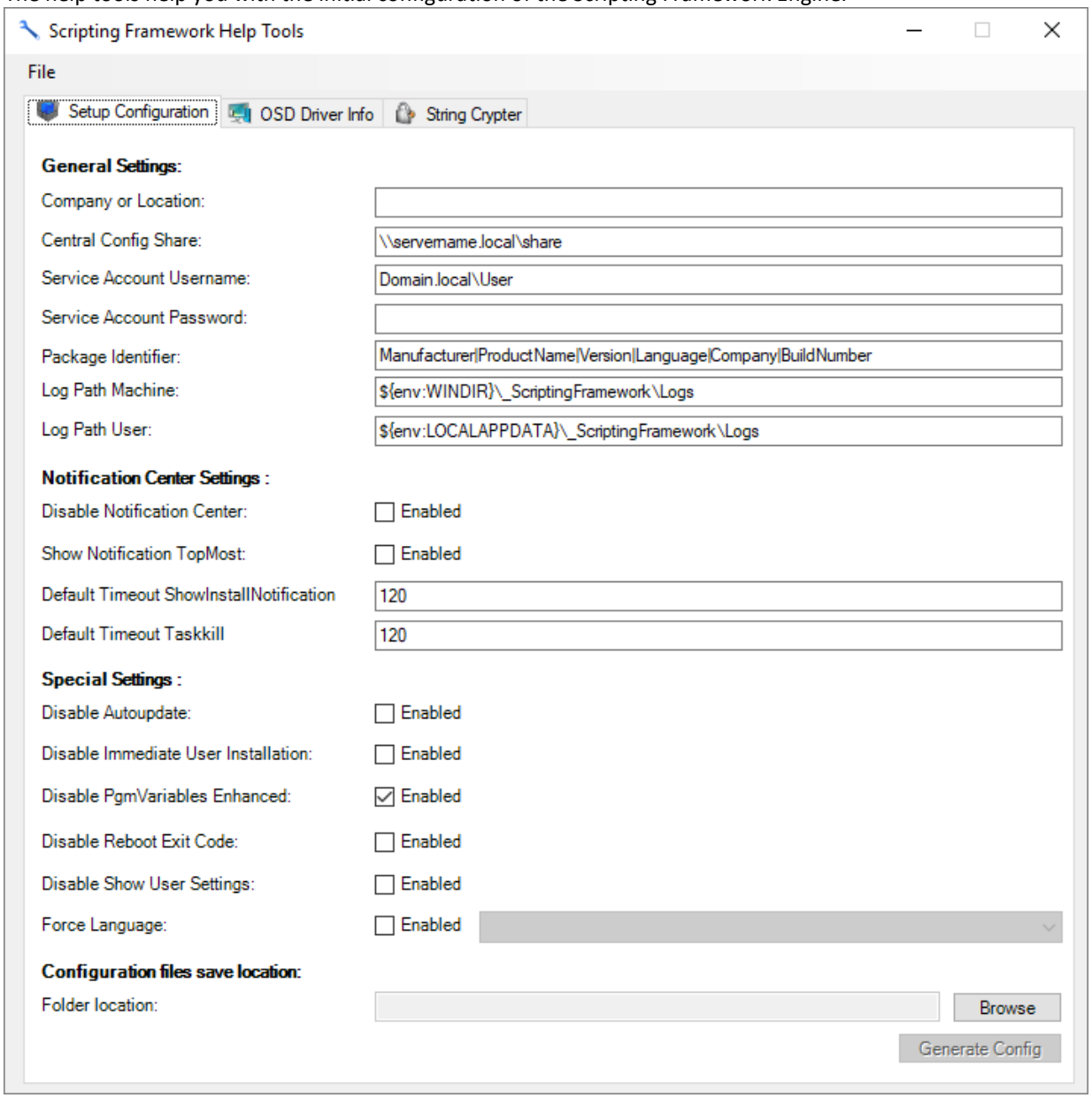
The result is a single executable file (NotepadLauncher.exe), which includes the additional text file:



By calling the NotepadLauncher.exe, Notepad starts with the integrated Text.txt. For the execution on the client is Scripting Framework needed.

3.2 Help Tools

The help tools help you with the initial configuration of the Scripting Framework Engine.



3.3 Manual English

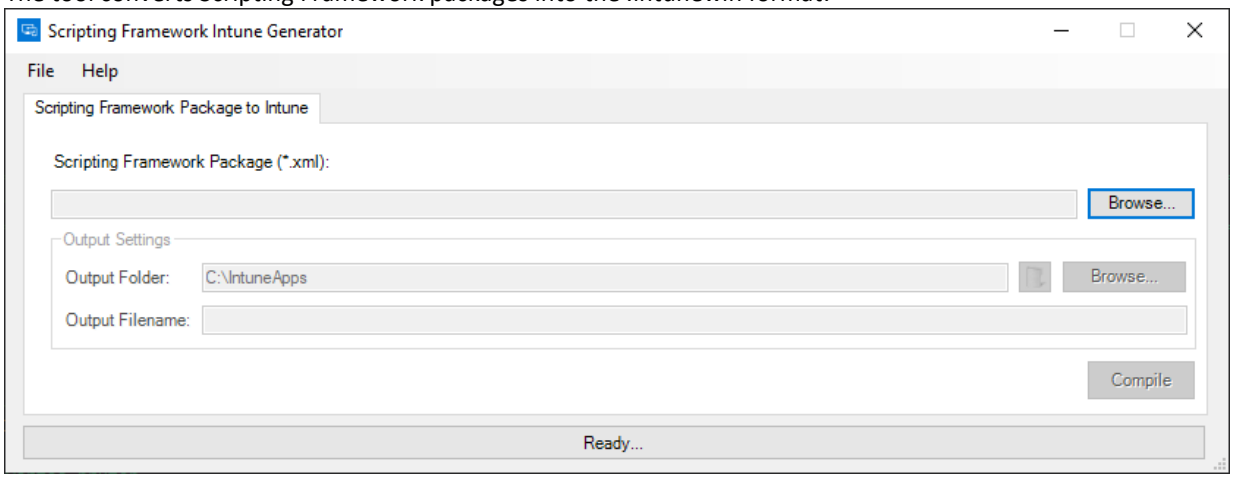
The Scripting Framework Manual in English as a PDF file.

3.4 Manual German

The Scripting Framework Manual in German as PDF file.

3.5 Intune Generator

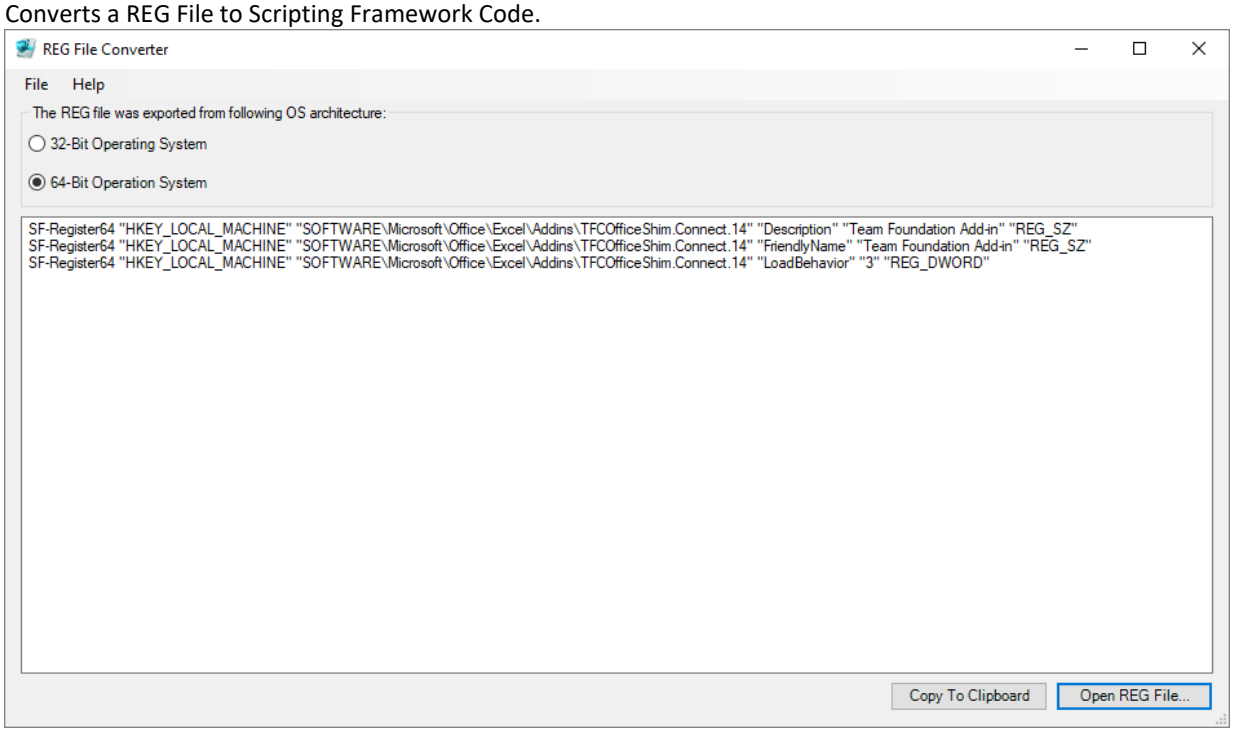
The tool converts Scripting Framework packages into the .intunewin format.



After the conversion, the package can be uploaded to Intune.

3.6 REG File Converter

Converts a REG File to Scripting Framework Code.

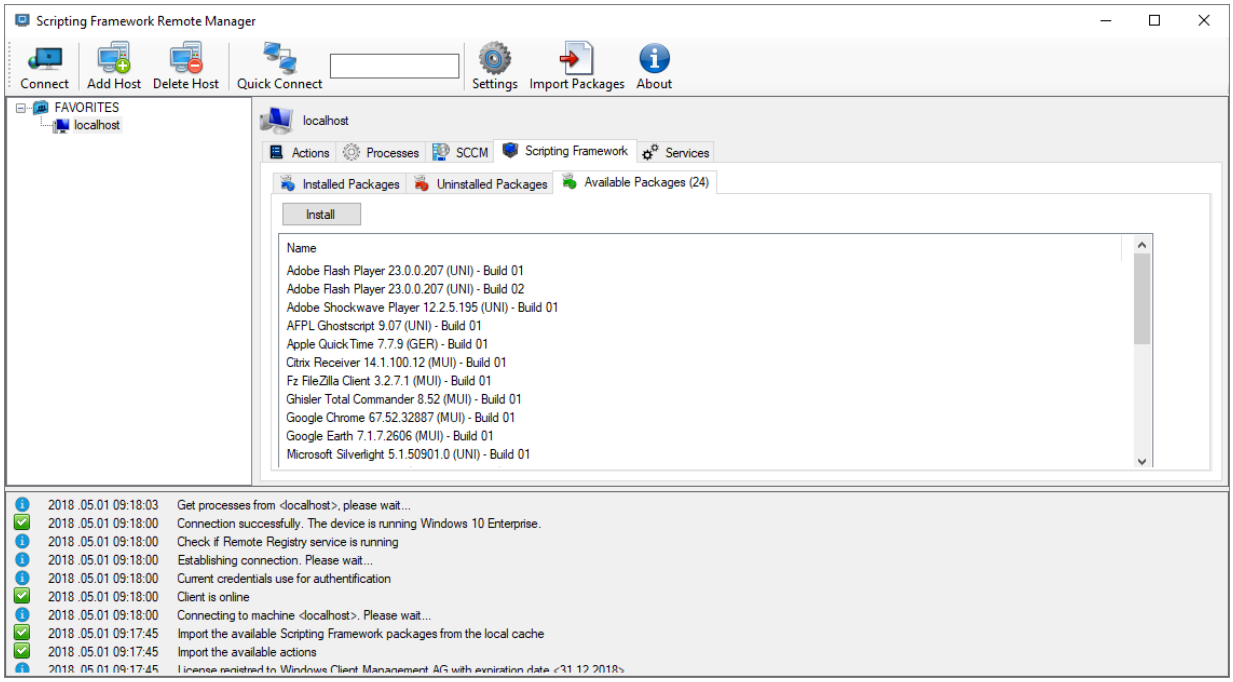


To convert a REG file, first select whether the file was exported from a 32-bit or 64-bit OS. Then drag and drop the desired REG file into the window, or use the "Open REG File..." button.

The REG file is then automatically converted. With the button "Copy To Clipboard" all content is copied to the clipboard. Individual lines can also be marked and copied. After copying to the clipboard you can paste the code in the desired package.

3.7 Remote Manager

With Remote Manager, Scripting Framework software packages can be manually installed and uninstalled on individual clients. At the same time, you have access to useful actions and information.



Requirements

- Microsoft .NET Framework 4.5.0 or higher
- Local administrator rights on the client to be connected
- The service *Remote Registry* must be started
- The firewall settings must be set so that the connection to the share admin $ is allowed

3.7.1 Tool Bar

[Connect]
Connect to a favorite.

[Add Host]
Add a favorite. Double-click on the corresponding favorite to connect directly to the client.

[Delete Host]
Remove a favorite.

[Quick Connect]
Clicking on the button or pressing the Enter key establishes a connection to the client, which was entered in the adjacent text box.

[Settings]
Open the available settings, such as the package share on which the packages are stored.

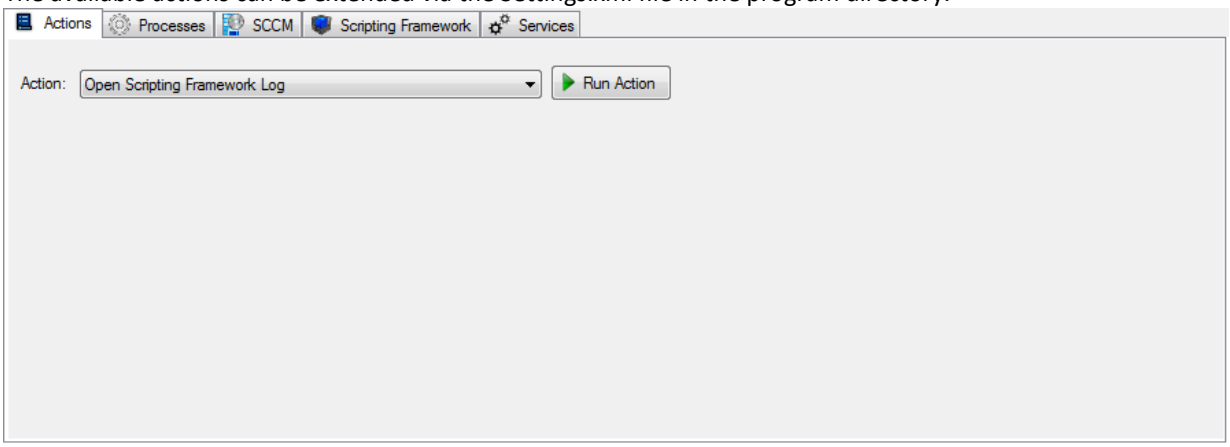
[Import Packages]
Import the scripting framework package information from your package share, which must be deposited accordingly in the settings. After the import process, you can install additional packages to the connected client or uninstall previously installed packages.

The package information is cached so that it is immediately available when the Remote Manager is restarted. If a package is added or removed, a new import is required.

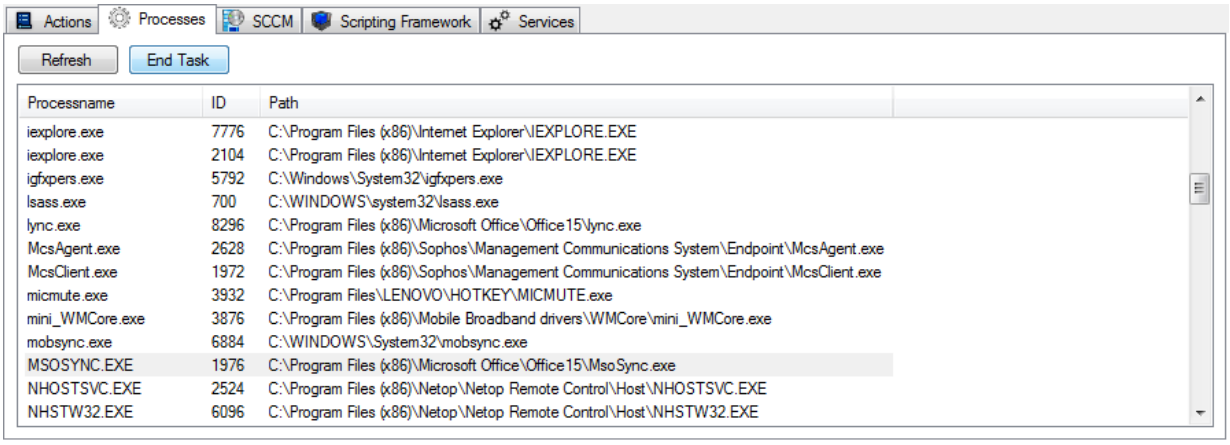
3.7.2 Tab Pages

[Actions]
Under Actions, you have the ability to perform frequently needed actions, such as Opening the Scripting Framework Log on the appropriate computer. To perform an action, you do not necessarily need to connect to the client with Remote Manager. It is sufficient if you select a client from the favorites (no double click), or you just enter the host name in the field Quick Connect (without pressing the Enter key or clicking on the Quick Connect button).

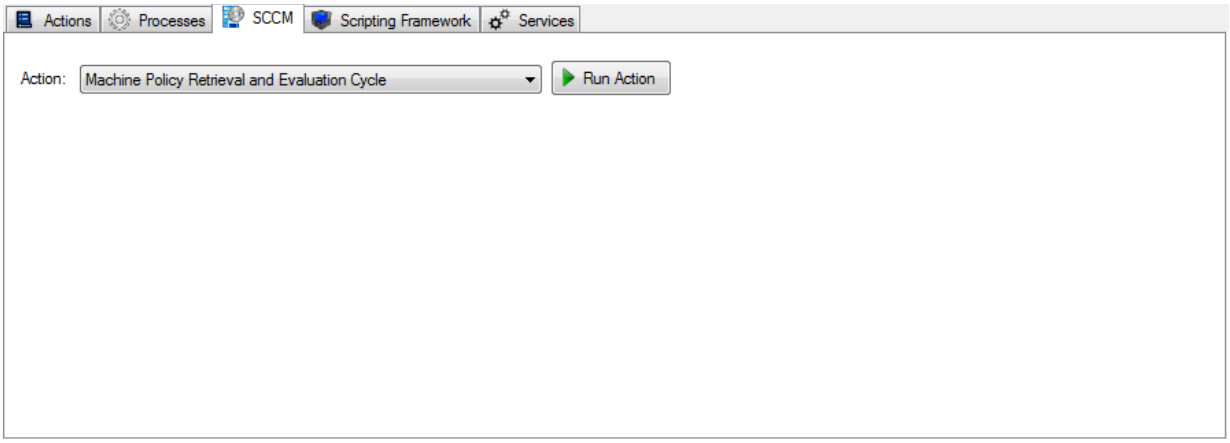
The available actions can be extended via the Settings.xml file in the program directory.



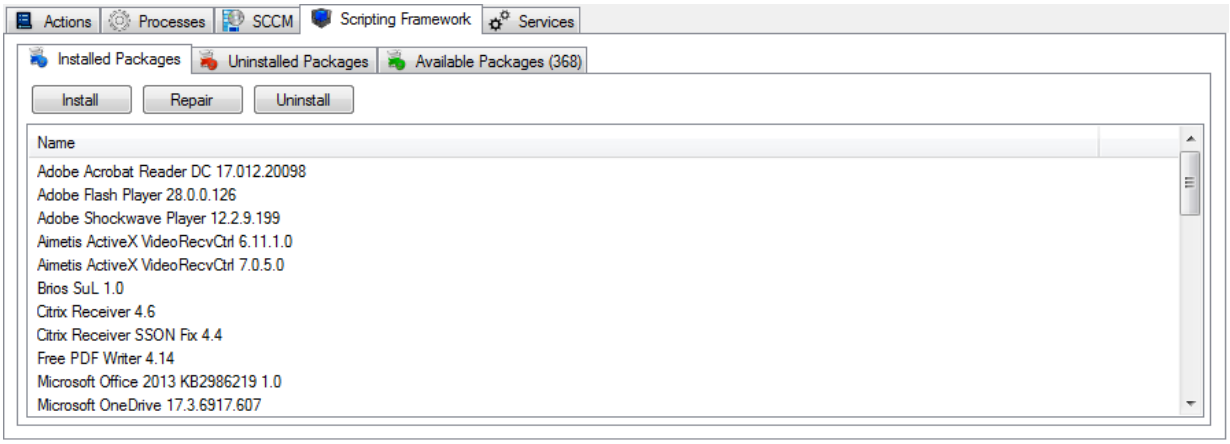
[Processes]
Under Processes, the currently running processes are displayed on the connected client with the option to terminate them.



[SCCM]
Allows starting the existing SCCM Client Actions on the connected client.



[Scripting Framework - Installed Packages]
Here are the packages that are installed on the connected client.



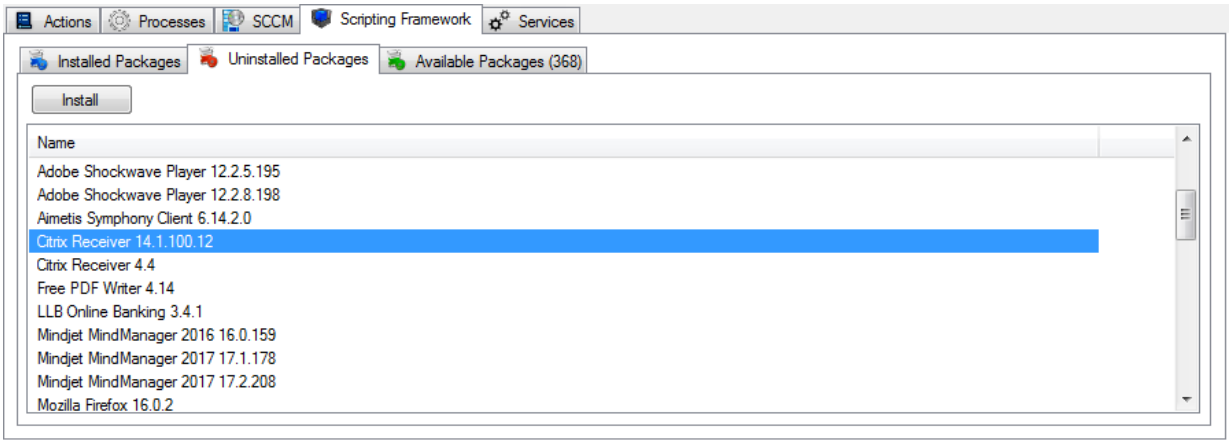
After selecting a package, the following three options are available via the button:

Install again
The package will be reinstalled again.

Repair
The package is first uninstalled and then reinstalled.

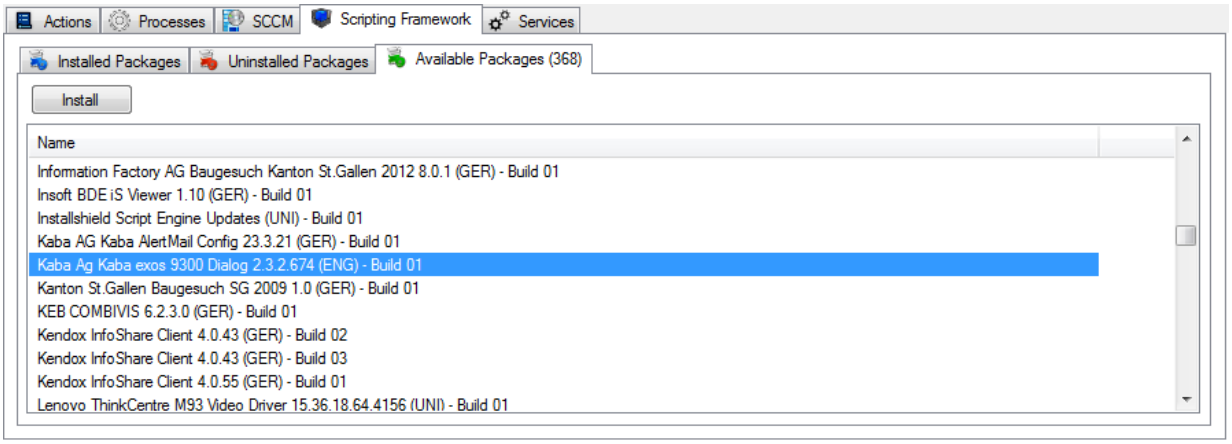
Uninstall
The package will be uninstalled.

[Scripting Framework - Uninstalled Packages]
A list of packages uninstalled on the device.

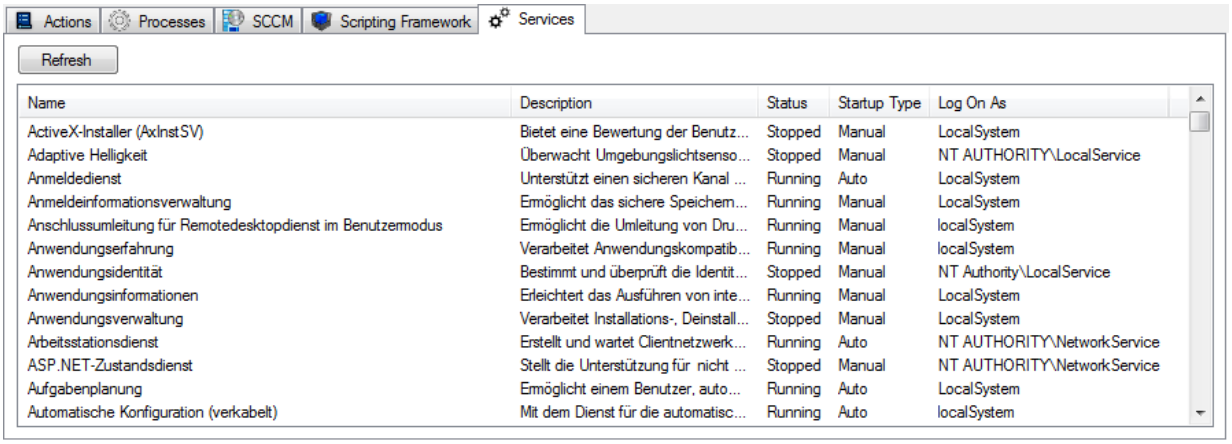


To reinstall a uninstalled package, select it and then click the *Install* button.

[Scripting Framework - Available Packages]
Here you can see all available Scripting Framework packages that you have imported before. Select the desired package and click the Install button to install the package.

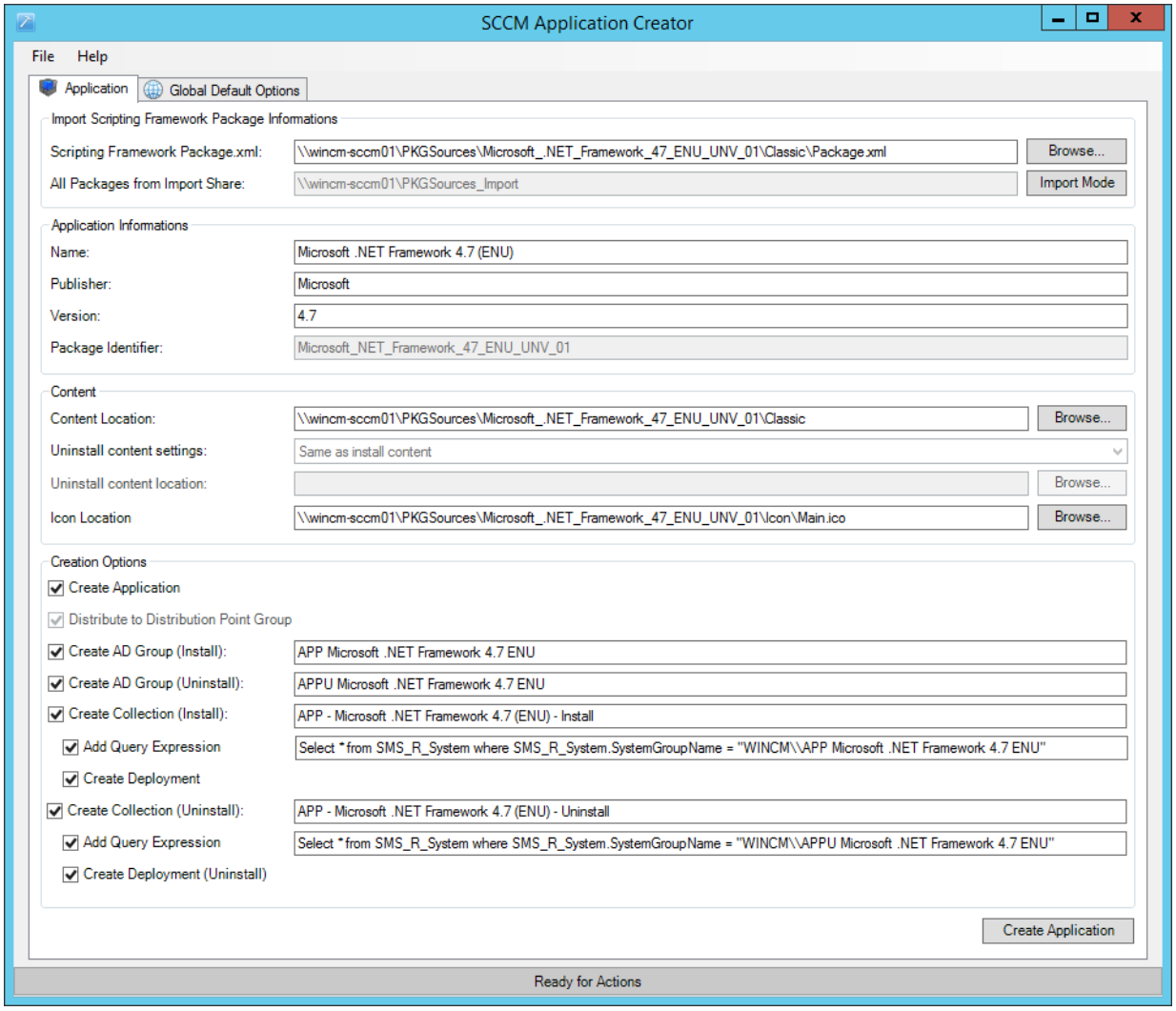


[Services]
A list of all services available on the connected device.



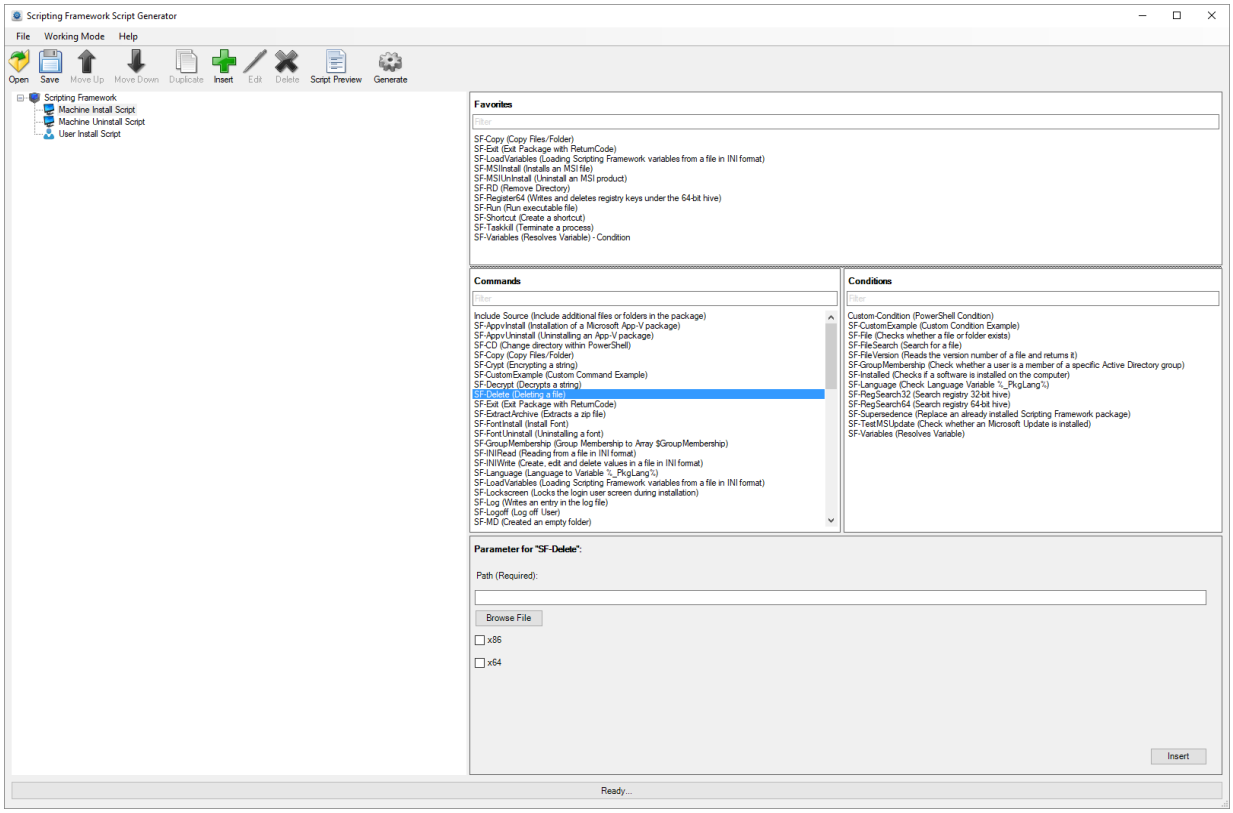
3.8 SCCM Application Creator

Quickly and efficiently create a Scripting Framework package in the SCCM as an application including collections, AD groups, etc. Creation is based on the settings you have defined in the "Global Default Options".



3.9 Script Generator

The Script Generator makes it easy to create the complete code with the functions of Scripting Framework. It is possible to edit the generated package at any time using the Script Generator.



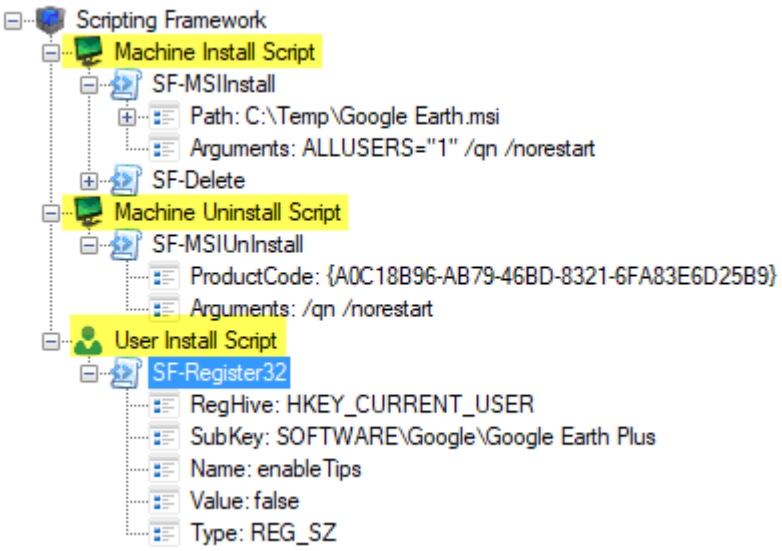
On the right side, you can see the available *Scripting Framework Functions*. These can be selected with a simple click. In the lower right area, the available parameters can be selected and supplemented.

By clicking on the *Insert* button, the function is moved to the left area, which depicts the actual script. This is divided into three areas *[Machine Install Script], [Machine Uninstall Script] and [User Install Script]*.

3.9.1 Script window

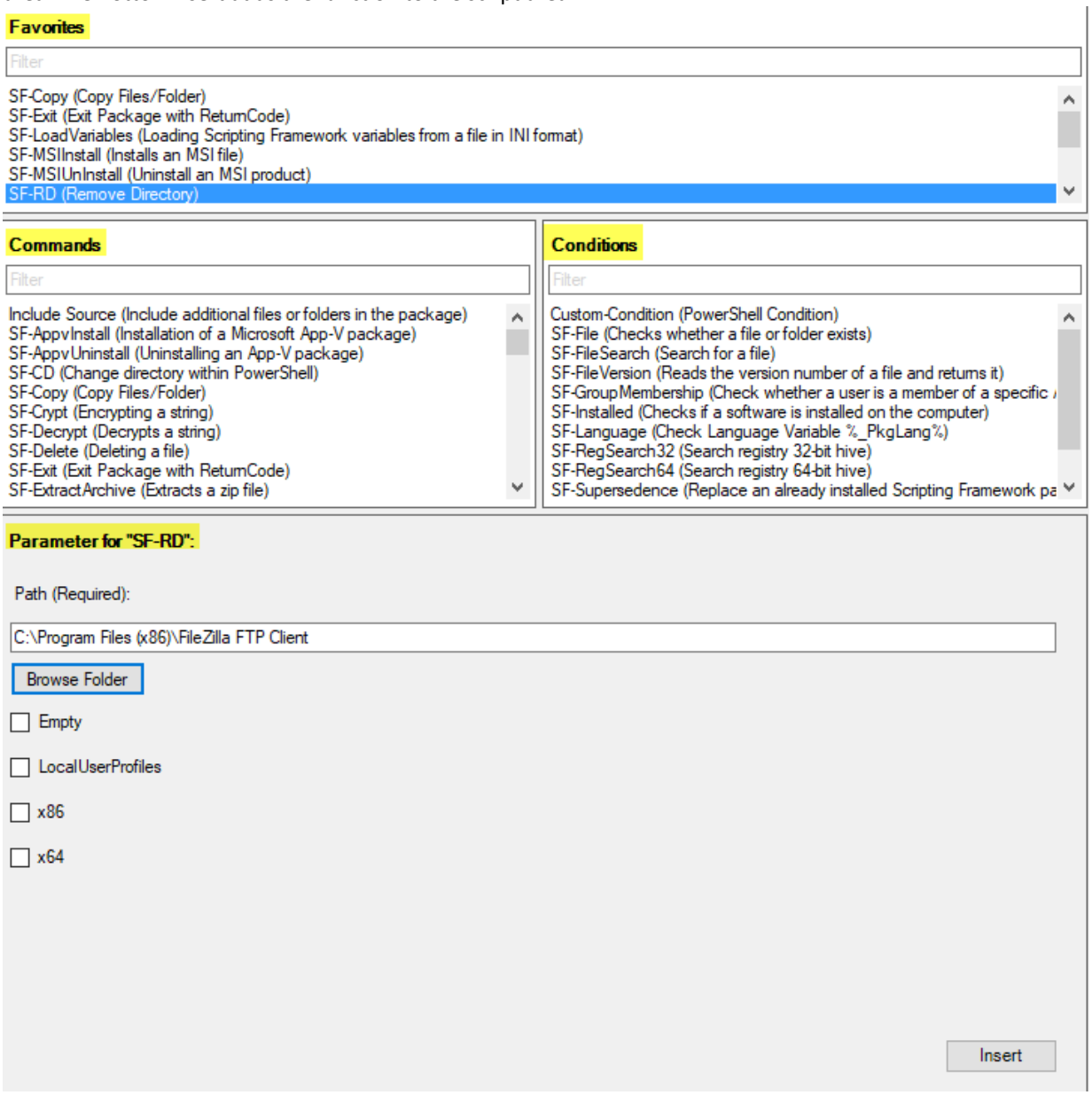
The script window has three areas:

- **Machine Install Script** -> entspricht Install.ps1
- **Machine Uninstall Script** -> entspricht Uninstall.ps1
- **User Install Script** -> entspricht InstallUser.ps1

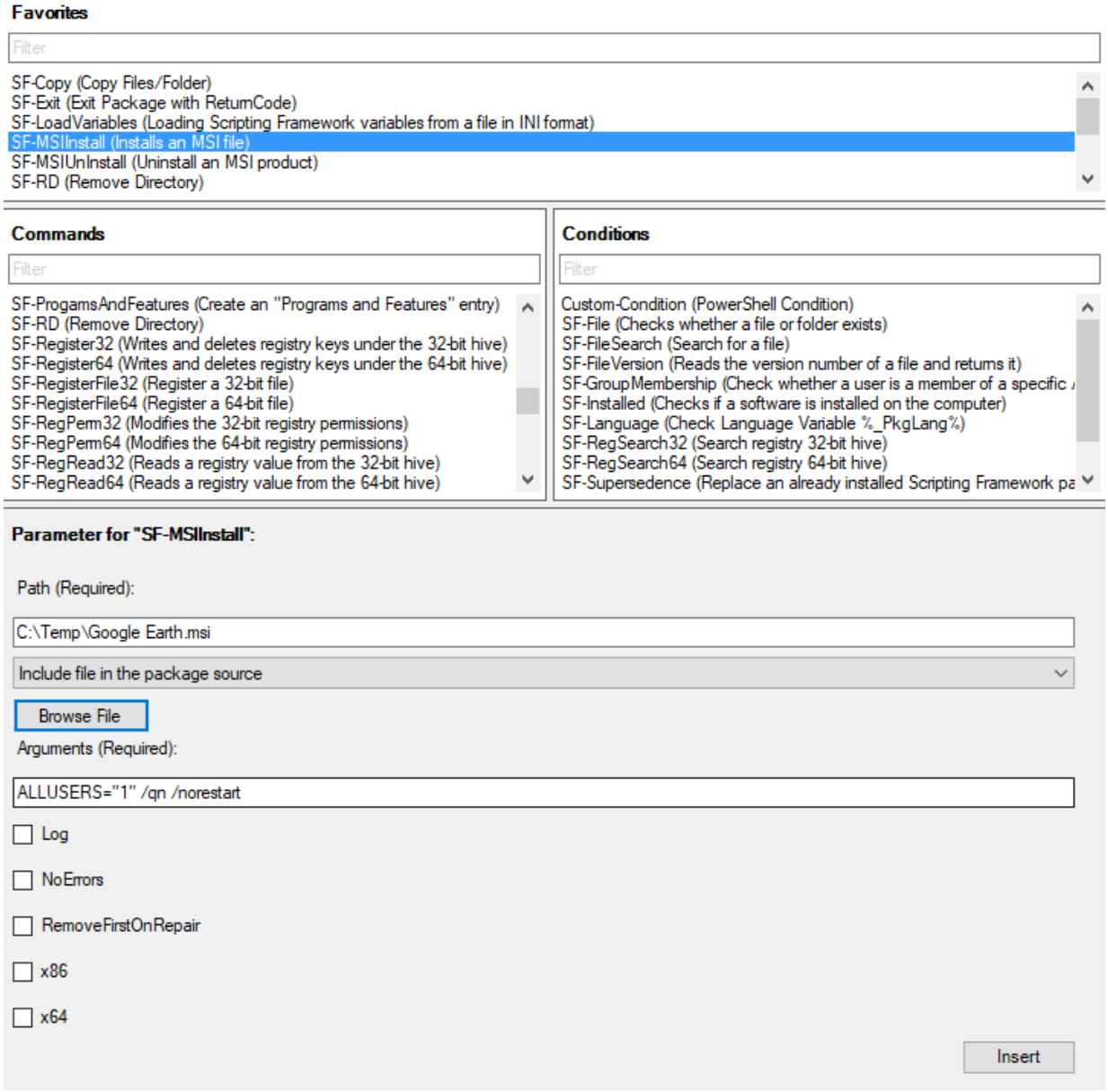


3.9.2 The functional area

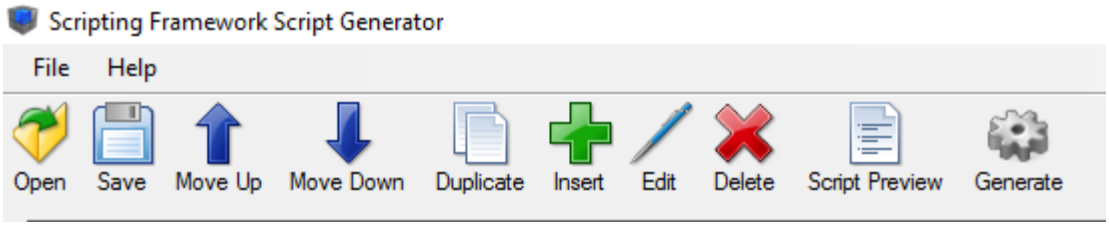
Here are all functions of Scripting Framework, as well as the possible conditions available.
When a function or condition is selected, the necessary and possible parameters become visible in the lower area. The Botton Insert adds the function to the script area.



Example „SF-MSIInstall“

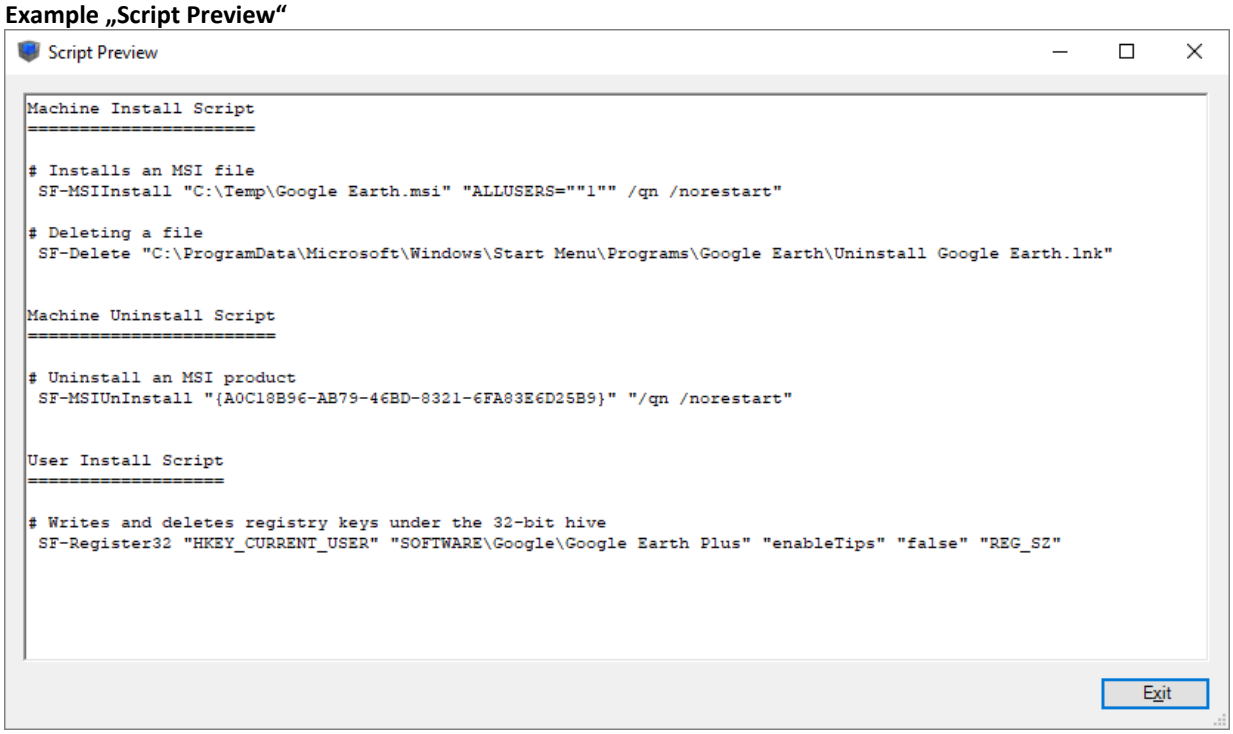


3.9.3 The menu bar

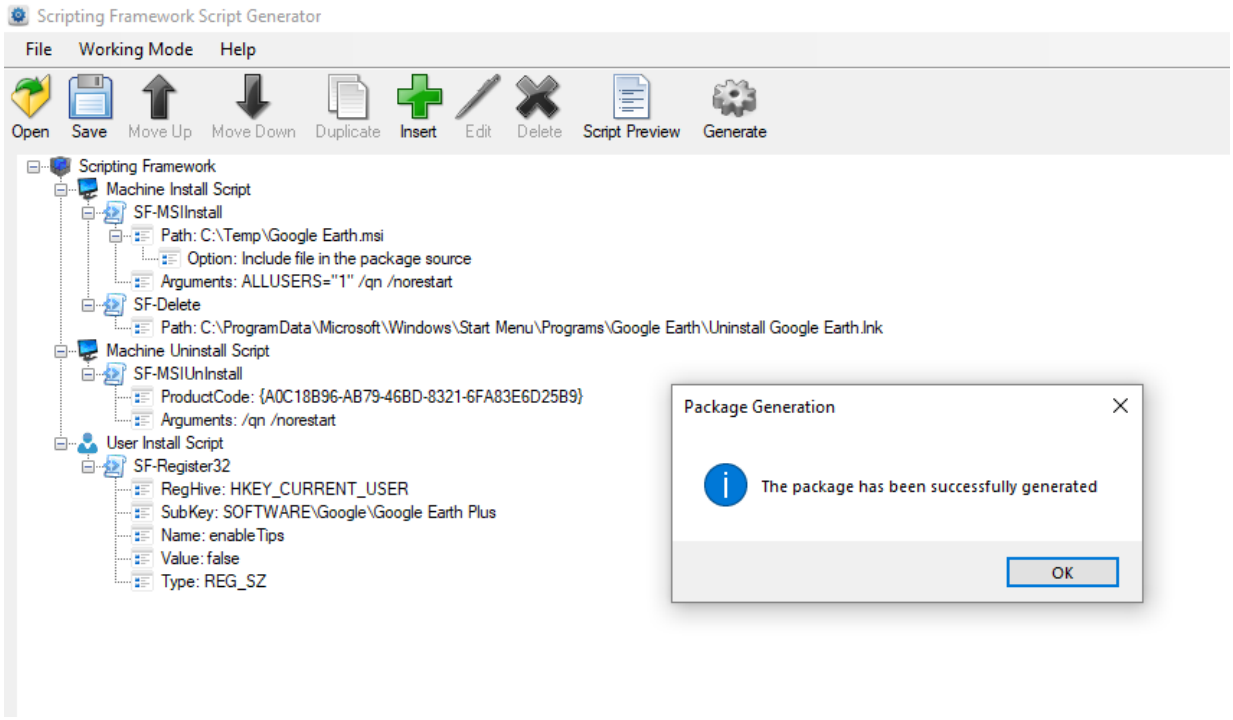


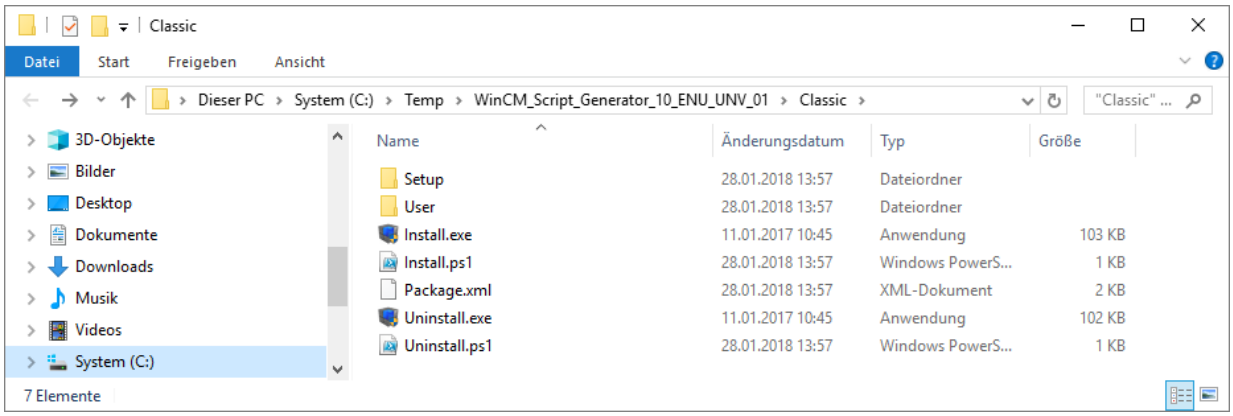
Open – Opens an existing package
Save – Saves the project
Move Up – Move the selected function up in the Script area
Move Down - Move the selected function down in the Script area
Duplicate – Duplicate a function / condition
Insert – Paste the selected function from the right window into the script area
Edit – Edit the selected function in the script area
Delete – Delete a function / condition
Script Preview – Displays the script as a Scripting Framework code
Generate – Generates the software package

Example „Script Preview“



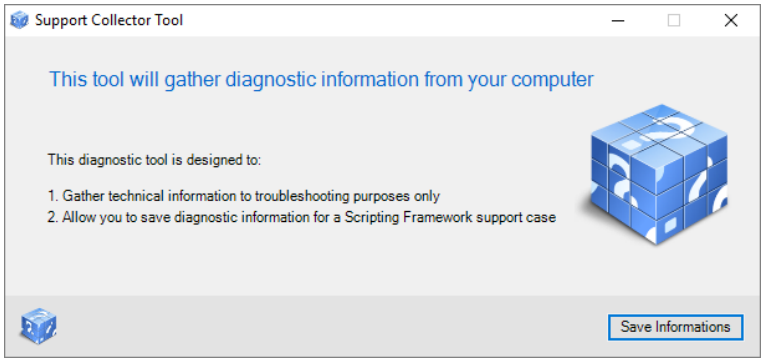
Example „Generate"



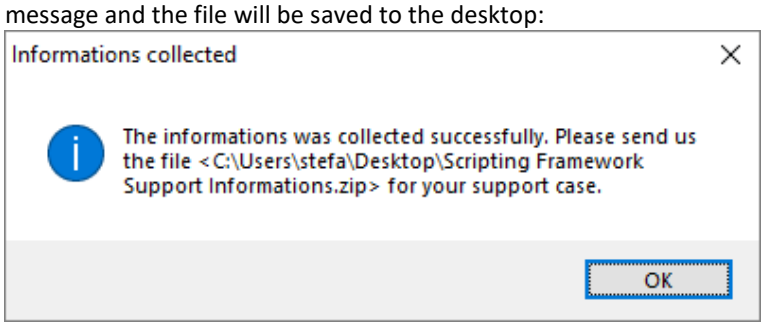


3.10 Support Collector Tool

The Support Collector Tool collects all important information and logs that are relevant for a support case.

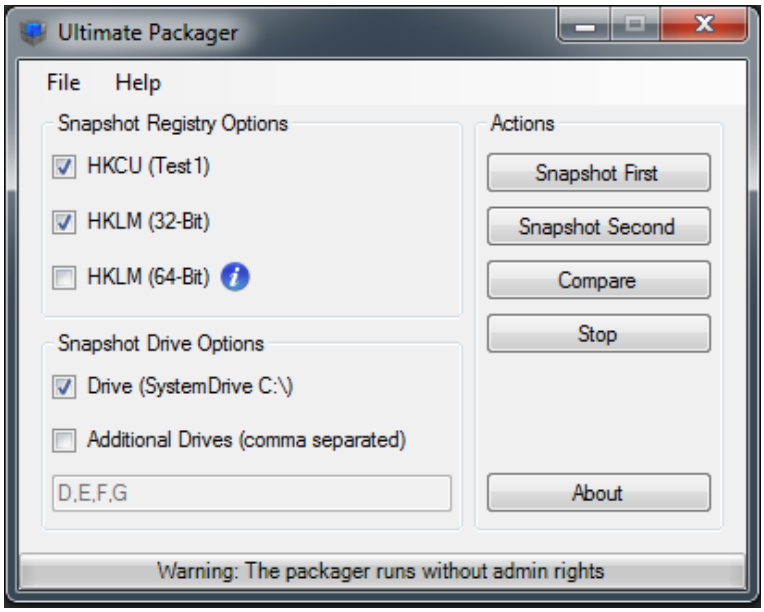


To collect the information, click on „Save Informations“. Once the process is complete, you will receive a message and the file will be saved to the desktop:



3.11 Ultimate Package

With the *Ultimate Packager*, you can quickly and easily create software packages based on Scripting Framework functions. The tool uses the snapshot method. Before and after installing an application, a snapshot is recorded. A fully functional package is then created from the changes.



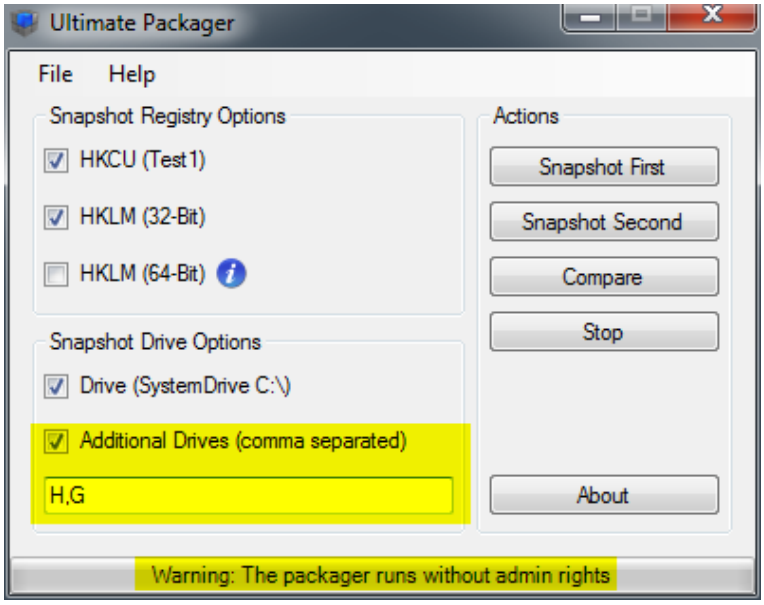
Use

1. Run *Snapshot First*
   Various settings are available. Select which areas of the registry are taken into account as well as with or without files, incl. additional drives (not just Drive C).
2. Perform the installation or adjust the client settings manually.
3. Run *Snapshot Second*
4. *Compare*
   Now the changes resulting from an installation or adjusted settings are recorded in a Scripting Framework package.

The package is created in the folder *C:\Scripting_Framework\Suite\Ultimate Packager\Output\Scripting Framework* and can be copied for further use.

Important Note
You can start the Ultimate Packager without Suite by running the file "*C:\Scripting_Framework\Suite\Ultimate Packager\UltimatePackager.exe*"

For "standard" use, start the *Ultimate Packager (Suite)* with local administrator rights.
However, if you are taking network drives such as the user home into account for the snapshot, please start the *Ultimate Packager (Suite)* with "normal" user rights.



4 Help and examples

4.1 Functions withing the script

Example to show all available commands for the engineer:
Get-Command SF-*

```
PS C:\WINDOWS\system32> get-command SF-*

CommandType     Name
-----------     ----
Function        SF-AppvInstall
Function        SF-AppvUninstall
Function        SF-CD
Function        SF-Copy
Function        SF-Crypt
Function        SF-Decrypt
Function        SF-Delete
Function        SF-Exit
Function        SF-File
Function        SF-FontInstall
Function        SF-FontUninstall
Function        SF-GroupMembership
Function        SF-INIRead
Function        SF-INIWrite
Function        SF-Installed
Function        SF-Language
Function        SF-LoadVariables
Function        SF-LockScreen
Function        SF-Log
Function        SF-MD
Function        SF-MSIInstall
Function        SF-MSIPatchInstall
Function        SF-MSIRepair
Function        SF-MSIUninstall
Function        SF-MSIUninstallByDisplayName
Function        SF-NTFSPerm
Function        SF-Path
Function        SF-PinnedApplication
Function        SF-RD
Function        SF-Register32
Function        SF-Register64
Function        SF-RegisterFile32
Function        SF-RegisterFile64
Function        SF-RegPerm32
Function        SF-RegPerm64
Function        SF-RegRead32
Function        SF-RegRead64
Function        SF-RegSearch32
Function        SF-RegSearch64
Function        SF-RemoveVariables
Function        SF-Rename
Function        SF-Replace
Function        SF-Run
Function        SF-Service
Function        SF-ServiceInstall
Function        SF-Set
Function        SF-Shortcut
Function        SF-ShowInstallNotification
Function        SF-SystemReboot
Function        SF-Taskkill
Function        SF-Textfile
Function        SF-UnRegisterFile32
Function        SF-UnRegisterFile64
Function        SF-Variables
Function        SF-Wait
Function        SF-WUSAInstall
```

Each function has a help and examples.

Example for help:
Get-Help SF-Exit

```
PS C:\WINDOWS\system32> Get-Help SF-Exit

NAME
    SF-Exit

ÜBERSICHT
    Exit the PowerShell script


SYNTAX
    SF-Exit [[-Message] <Object>] [<CommonParameters>]


BESCHREIBUNG
    Write a log entry and closes the PowerShell Script with the ExitCode 1603


VERWANDTE LINKS


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

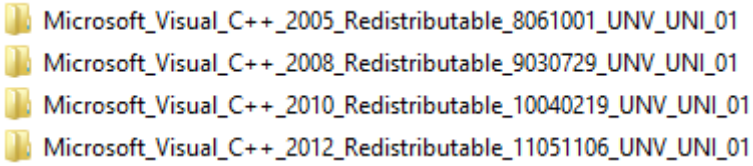
Example to show Examples:
Get-Help SF-Exit -Examples

```
PS C:\WINDOWS\system32> Get-Help SF-Exit -Examples

NAME
    SF-Exit

ÜBERSICHT
    Exit the PowerShell script

    -------------------------- BEISPIEL 1 --------------------------

    PS C:\>SF-Exit "My Example Message"


    -----------
    Description
    Exit the Script with the log entry "My Example Message"
```

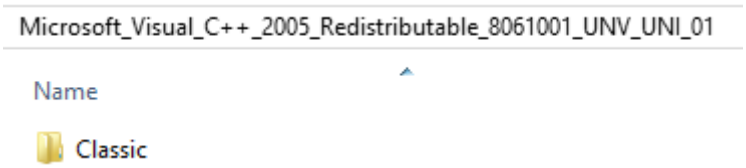
5 Scripting Framework software package (Definition)

5.1 Package name (main folder)

Each packet is stored in a separate folder:



By this we recommend once a subdivision, but this is not a requirement. We distinguish between classic and AppV:



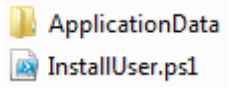
Classic corresponds to a classical package (MSI, Silent Installation, snapshot) and AppV to a Microsoft AppV package.

5.2 Folder and file structure

The main structure of a package under "Classic" or "AppV" looks as follows:

Config Setup User Install.exe Install.ps1 Package.xml Uninstall.exe Uninstall.ps1	**Config**	Configuration file which belongs to the package (.cfg)
	Setup	Source of the setup files (MSI, Silent Setups, etc.)
	User	User-specific files for user settings.
	Install.exe	The file is used for the installation of the package.
	Install.ps1	This file is the logic of the package (Installation).
	Package.xml	Definition of the package (manufacturer, product name, version, etc.).
	Uninstall.exe	The file is for the deinstallation of the package.
	Uninstall.ps1	This file is the logic of the package (uninstall).

5.2.1 User structure - In the folder „User“

	**ApplicationData**	As an example the Application Data folder can be copied to %_ApplicationData%, with using the Install User.ps1
	InstallUser.ps1 oder Disabled_InstallUser.ps1	The file contains the logic for user-specific operations (e.g., the writing of a registry key, the copying of files, etc.).

If the file "User\InstallUser.ps1“ exists, an entry for the user settings in the registry is automatically set during the installation and the user-specific content is transferred to "C:\Windows_Scripting Framework\Cache". Scripting Framework regulates the entire process automatically, there are no further actions by the packager necessary. In order to deactivate the user part, the file must be renamed ("Disabled_InstallUser.ps1"). Scripting Framework requires a file, do not delete them.

5.3 Details of the individual files

5.3.1 Install.exe

Comment:
Start Scripting Framework and processes the install.ps1. Then the EXE with the corresponding return code is terminated.

5.3.2 Install.ps1

Comment:
Contains the installation script. An example of an MSI installation:

Content:

```
# ================================================================================================
# Installation
# ================================================================================================

# Terminate Process
 SF-Taskkill "AcroRd32.exe"

# Language
  SF-Language "1031,1033,1036,1040"

# Installation Reader
 SF-MSIInstall "%_PkgSource%\Setup\AcroRead.msi"
"TRANSFORMS='%_PkgSource%\Setup\%_PkgLang%.mst' ALLUSERS='1' DISABLEDESKTOPSHORTCUT='1'
EULA_ACCEPT='YES' SUPPRESS_APP_LAUNCH='YES' /qn /norestart"

# Installation Language Packs
 SF-MSIInstall "%_PkgSource%\Setup\FontPack11000_XtdAlSF-Lang.msi" "ALLUSERS='1' /qn
/norestart"

# Installation Patches
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrUpd11001_MUI.msp" "ALLUSERS='1' /qn
/norestart"
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrSecUpd11002.msp" "ALLUSERS='1' /qn
/norestart"
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrUpd11003_MUI.msp" "ALLUSERS='1' /qn
/norestart"
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrUpd11004_MUI.msp" "ALLUSERS='1' /qn
/norestart"
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrSecUpd11005.msp" "ALLUSERS='1' /qn
/norestart"
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrUpd11006_MUI.msp" "ALLUSERS='1' /qn
/norestart"
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrUpd11007_MUI.msp" "ALLUSERS='1' /qn
/norestart"
```

```
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrSecUpd11008.msp" "ALLUSERS='1' /qn
/norestart"
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrUpd11009_MUI.msp" "ALLUSERS='1' /qn
/norestart"
 SF-MSIPatchInstall "%_PkgSource%\Setup\Patch\AdbeRdrUpd11010_MUI.msp" "ALLUSERS='1' /qn
/norestart"

# Delete Registry
 SF-Register32 "HKey_Local_Machine" "SOFTWARE\Microsoft\Windows\CurrentVersion\Run" "Adobe
Reader Speed Launcher" "{nil}" "REG_SZ"
 SF-Register32 "HKey_Local_Machine" "SOFTWARE\Microsoft\Windows\CurrentVersion\Run" "Adobe ARM"
"{nil}" "REG_SZ"
 SF-Register32 "HKey_Local_Machine" "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
"AdobeAAMUpdater-1.0" "{nil}" "REG_SZ"

# Disable and Stop Service
 SF-Service "AdobeARMservice" -Stop
 SF-Service "AdobeARMservice" -Disabled

# Modify Registry
 SF-Register32 "Hkey_Local_Machine" "SOFTWARE\Adobe\Acrobat Reader\11.0\AdobeViewer" "EULA" "1"
"REG_DWORD"
 SF-Register32 "Hkey_Local_Machine" "SOFTWARE\Adobe\Acrobat Reader\11.0\AdobeViewer" "Launched"
"1" "REG_DWORD"
 SF-Register32 "Hkey_Local_Machine" "Software\Adobe\Acrobat Reader\11.0\Workflows"
"bEnableAcrobatHS" "0" "REG_DWORD"
 SF-Register32 "Hkey_Local_Machine" "Software\Adobe\Acrobat Reader\11.0\Workflows"
"bEnableShareFile" "0" "REG_DWORD"
 SF-Register32 "Hkey_Local_Machine" "Software\Adobe\Acrobat Reader\11.0\Workflows"
"bEnableRTCAuth" "0" "REG_DWORD"

# Delete Updater
 SF-Delete "%_ProgramFiles32%\Adobe\Reader 11.0\Reader\plug_ins\Updater.api"
 SF-Register32 "Hkey_Local_Machine" "SOFTWARE\Adobe\Acrobat Reader\11.0\FeatureLockDown"
"bUpdater" "0" "REG_DWORD"
 SF-Register32 "Hkey_Local_Machine" "SOFTWARE\Adobe\Acrobat Reader\11.0\FeatureLockDown"
"bCreatePDFOnline" "0" "REG_DWORD"
 SF-Register32 "Hkey_Local_Machine" "SOFTWARE\Microsoft\Windows\CurrentVersion\Run" "Adobe ARM"
"{NIL}" "REG_SZ"
 SF-Register32 "Hkey_Local_Machine" "SOFTWARE\Adobe\Adobe ARM\1.0\ARM" "iCheck" "0" "REG_DWORD"

# Disable Product Improvement Program
 SF-Register32 "Hkey_Local_Machine" "SOFTWARE\Policies\Adobe\Acrobat
Reader\11.0\FeatureLockDown" "bUsageMeasurement" "0" "REG_DWORD"

# ================================================================================================
# END
# ================================================================================================
```

5.3.3 Package.xml

Comment:
In the XML will be the most important information about the package. The information is used for various tasks in the Scripting Framework uses. For example, for MSI log files, config files, Package identifier, etc.

To define values in XML:

Name:	Value:
BuildNumber	Corresponds to the build version of the package. This is required for the distinction, if the packet already exists in the identical version. This is generally carried out at an expansion of the package or any other change in the case.
Company	Specifying whether the packet is intended for a specific company or site. The value can be freely defined. For a package that can be used independently, we recommend the acronym UNV for "universal". If a package containing specific settings for a site / company, we generally use a three-letter abbreviation. For example, Zurich ZHR.
Manufacturer	The manufacturer of the software
ProductName	The software product name
Version	The exact version of the software
VersionShort	Main version number
Language	The language of the software. We recommend the naming convention from Microsoft. Examples: German = DEU French= FRA Italian = ITA English = ENU Multilanguage = MUI Universal = UNI → for example, a configuration package, which sets settings and has no actual language.

Structure of the XML File (Content):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
 # ================================================================================================
 # (c) Copyright by Windows Client Management - www.wincm.ch
 # ================================================================================================
 #
 # Package created :   01.01.2015 by : Windows Client Management AG
 # Package changed : 00.00.0000 by :
 #
 # ================================================================================================

 # ================================================================================================
 # Prerequisites:
 # - None
 # ================================================================================================
-->

<!-- Package Definition -->
<Package>
        <BuildNumber>01</BuildNumber>
        <Company>UNV</Company>
        <Manufacturer>Adobe</Manufacturer>
        <ProductName>Reader</ProductName>
        <Version>11.0.10</Version>
        <VersionShort>11</VersionShort>
        <Language>MUI</Language>
</Package>
```

5.3.4 Uninstall.exe

Comment:
Starts Scripting Framework and processes the Uninstall.ps1. Then the EXE is terminated with the appropriate return code.

5.3.5 Uninstall.ps1

Comment:
Includes the installation script. An example of an MSI uninstall:

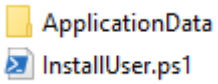
Content:
```
# ================================================================================================
# Uninstallation
# ================================================================================================

# Terminate Process
 SF-Taskkill "AcroRd32.exe"

# MSI Uninstall Adober Reader
 SF-MSIUninstall "{AC76BA86-7AD7-FFFF-7B44-AB0000000001}" "/qn /norestart"

# MSI Uninstall Software Reader FontPack
 SF-MSIUninstall "{AC76BA86-7AD7-2530-0000-A00000000004}" "/qn /norestart"


# ================================================================================================
# END
# ================================================================================================
```

5.3.6 InstallUser.ps1

Comment:
Includes the user configuration. Example:

Content:
```
# ================================================================================================
# Install User
# ================================================================================================

# Copy Files
 SF-Copy "%_PkgCache%\User\ApplicationData" "%_ApplicationData%"

# Registry write
 SF-Register32 "HKey_Current_User" "Software\Adobe\Acrobat Reader\11.0\AdobeViewer" "EULA" "1"
"REG_DWORD"
 SF-Register32 "HKey_Current_User" "Software\Adobe\Acrobat Reader\11.0\AVGeneral"
"bBrowserDisplayInReadMode" "0" "REG_DWORD"

# ================================================================================================
# END
# ================================================================================================
```

Filestructure (Classic\User):



6 Scripting Mode

All functions of Scripting Framework can be used outside by the "Scripting Mode" of a package sleeve. The scope of these are general scripts, Launcher or configurations within an App-V package.

To execute a script in the "Scripting Mode" create any .ps1 file is. Do not use names like Install.ps1, Uninstall.ps1 or InstallUser.ps1:

```
# ================================================================================
# Scripting Framework Script
# ================================================================================

# Example Parameter
SF-Log "$Parameter1"

# Example (Start Notepad)
SF-Run "%_Windows%\notepad.exe" "$Parameter1" -Wait -NoErrors -Show

# ================================================================================
# END
# ================================================================================
```

The .ps1 file can be run with the following CMD command:
"%SystemRoot%_ScriptingFramework\Modul\ScriptingFramework.exe" -File "C:\Temp\ScriptingModeExample.ps1" -Parameter1 "%WINDIR%\WindowsUpdate.log"

In this example, the Parameter1 passed the value C:\Windows\WindowsUpdate.log to the script, which is used to run the Notepad application. It can be passed any number of parameters to the script. The parameter name corresponds to the name of the PowerShell variable.

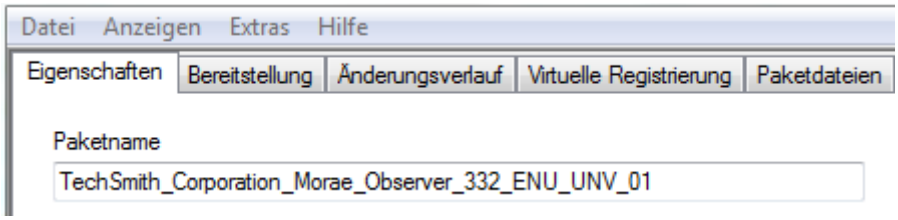
6.1 Application - Launcher

To write configurations before starting an application repetitively, the use of so-called launchers is possible. The original shortcut of the application will be bent to the launcher and the launcher script sets the desired settings and starts the actual application. Even in this scenario, it is possible to load dynamic variables with the function SF-LoadVariables from a CFG and use these for actions. Due to the lack of XML file that is within a launcher, the PkgIdentifier is not known. The PkgIdentifier is set to the value *Scripting_CurrentProccessID.* However, the identifier can be passed via command line parameter -PkgIdentifier and transferred to the Scripting Mode, which is also the variable %_PkgCFGName% automatically set correctly. You can download an example here.

Within the launcher, all functions can be used normally, for administrative tasks of the executing user must have the appropriate privileges.

6.2 Application - App-V

When applying the Scripting Mode within an App-V package, corresponding to the variable %_PkgCFGName% the App-V Package Name:



The Command *SF-LoadVariables* can be characterized in an App-V package similar as in a classic package apply. Within a script for App-V, available as usual all Scripting Framework functions.

Example XML for a call from an App-V Package:

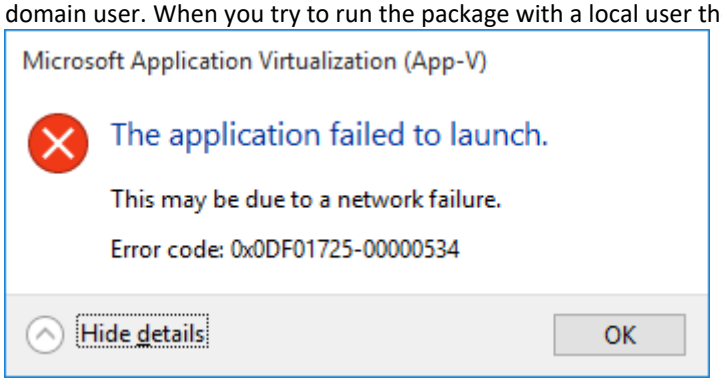
```
<!-- User Scripts -->
<UserScripts>
 <StartProcess RunInVirtualEnvironment="true">
  <Path>"C:\Windows\_ScriptingFramework\Modul\ScriptingFramework.exe"</Path>
  <Arguments>-File "[{AppVPackageRoot}]\..\Scripts\Launcher.ps1"</Arguments>
  <Wait RollbackOnError="true" />
  <ApplicationId>[{AppVPackageRoot}]\MoraeObserver.exe</ApplicationId>
 </StartProcess>
</UserScripts>
```

In order within a virtual App-V environments to address the root is the Scripting Framework variable *%_AppVPackageRoot%* available. Likewise, the use of the PowerShell variable *$AppVPackageRoot* is possible.

Important: The App-V client needs to allow the running of scripts. The configuration can be set once per PowerShell or GPO. PowerShell Command:

```
Set-AppvClientConfiguration -EnablePackageScripts $true
```

Up to and including the App-V client can run 5.1.86.0 App-V packages with integrated scripts only under a domain user. When you try to run the package with a local user the application appears an error message:

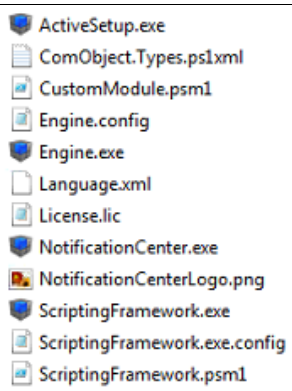


We hope that Microsoft will fix the problem with a hotfix.

7 Structure of the toolkit and further information

7.1 Files

The following files are required by the Scripting Framework and will be installed to "C:\Windows_ScriptingFramework":



Content „Modul_x86“ and „Modul_x64“ Folder

ActiveSetup.exe
Processes the pending Scripting Framework user settings.

ComObject.Types.ps1xml
The file is an XML-based text file, with the objects of Windows PowerShell, where you added properties and methods. This is internally required.

CustomModule.psm1
Includes your own Scripting Framework functions. For Details read the custom module manual.

Engine.exe | Engine.config
Contains the Scripting Framework Engine.

Language.xml
Translation texts for Notification Center.

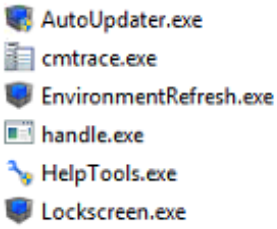
License.lic
It contains the license information which you receive from us.

NotificationCenter.exe
Shows the user the corresponding notifications, if a corresponding function is used.

NotificationCenterLogo.png
The Notification Center Logo, you can replace it with your own picture (480x50)

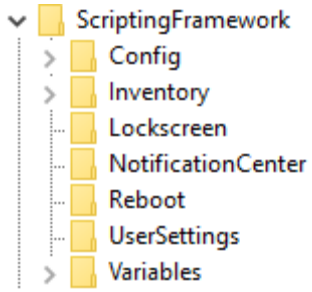
ScriptingFramework.exe | ScriptingFramework.exe.config
Main file that handles the call to the engine.

ScriptingFramework.psm1
Reduced PowerShell script which for the development (Engineering) of packages is needed.

	**Content „Tools“ Folder** **AutoUpdate.exe** Starts Autoupdate functionality. **cmtrace.exe** To view the log file **EnvironmentRefresh.exe** Updates the user environment variables. The tool is automatically addressed by Scripting Framework. **Handle.exe** Tool from Sysinternals (Microsoft) to terminate open handles. **HelpTools.exe** The Help tool of Scripting Framework, which create the configuration for the installation. **Lockscreen.exe** To lock the user's screen, this is used for SF-Lockscreen function.

7.2 Registry

Under „HKLM\Software*Wow6432Node*\ScriptingFramework“ you will found the following structure:



Config	Configuration Parameter of Scripting Framework.
Inventory	After the successful installation of a package, the information is stored under Inventory.
NotificationCenter	Internal entrys for the Notifcation Center.
Reboot	Intermediate storage of the Reboot Flags.
UserSettings	Information about the packages with user settings.
Variables	All relevant Scripting Framework Variables, which can be used within a script

7.2.1 Config

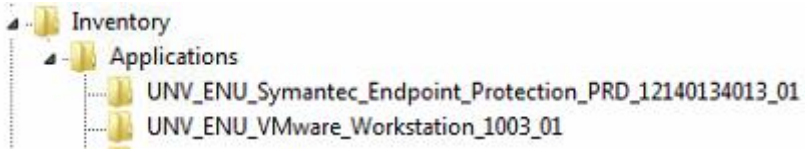
The following configuration values exists:

Name:	Value:	Description
DisableAutoupdate	0 or 1	Disables the automatic update of Scripting Framework.
DisableImmediateUserInstallation	0 or 1	Disables the automatic execution of the user settings (InstallUser.ps1) after the package installation.
DisableIsRegistryPOLGood	0 or 1	By default SF checks for a corrupt Registry.pol file and fixes it, as this will cause SF to crash. The value 1 can be used to disable the check and correction.
DisableNotificationCenter	0 or 1	With the value 1, you can deactivate the completely Notification Center. The Parameter -Notification will be ignored.
DisableRebootExitCode	0 or 1	If a Scripting Framework package request a reboot, will be return an return code 0 instead of 3010. This is usually only in the Citrix environment activated.
DisableShowUserSettings	0 or 1	No visible window during the installation of user settings.
ForceMachineLanguage	1031 1033 1036 1040	Fixed the language for the function SF-Language. This is normally only required on a language-specific Citrix.

LogPathMachine	${env:WINDIR}_ScriptingFramework\Logs	Pfad der Scripting Framework Log Dateien (Machine Actions)
LogPathUser	${env:LOCALAPPDATA}_ScriptingFramework\Logs	Pfad der Scripting Framework Log Dateien (User Actions)
NotificationTimeoutShowInstallNotification	0 or x	The Notification Timeout (SF-ShowInstallNotification) in seconds, provided that no further displacement for the installation is possible. After this timeout (displays as a countdown for the user), the installation continues. With the value 0, the automatic time-out is disabled, but this is not the recommendation.
NotificationTimeoutTaskkill	0 or x	Timeout in seconds until the tasks are automatically closed. With the value 0, the automatic timeout is disabled, but not the recommendation.
NotificationTopMost	0 or 1	The Notification Center is permanently displayed in the foreground. 0=Off 1=On
Password	**********	Password of the service account for the Share Access
PkgIdentifier	Manufacturer|ProductName|Version|Language|Company|BuildNumber	This value is determined PkgIdentifier
Username	Domain.local\User	The user name of the service account for the Share Access

7.2.2 Inventory

Following the installation of a package, the information are stored in the application inventory:



You find for every Scripting Framework package the following details:

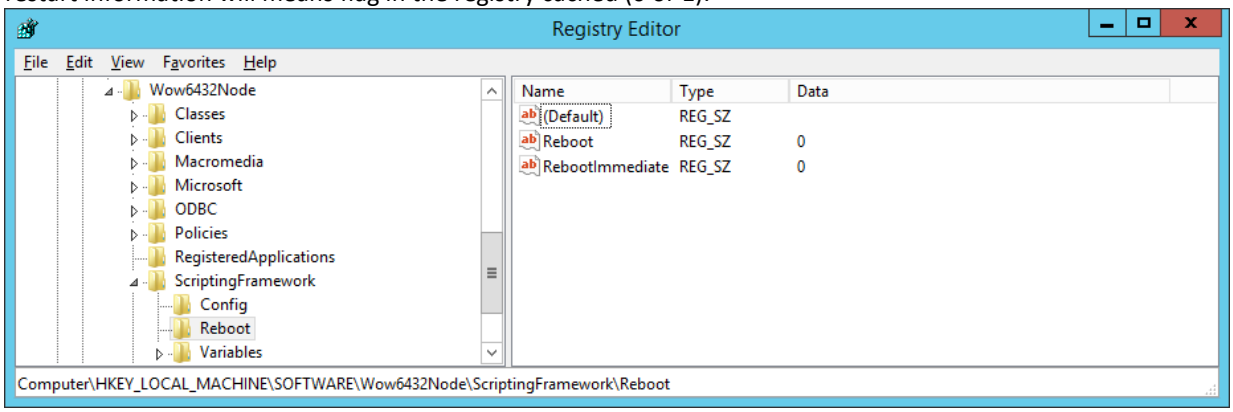
Name	Typ	Daten
(Standard)	REG_SZ	(Wert nicht festgelegt)
BuildNumber	REG_SZ	01
Company	REG_SZ	UNV
DisplayName	REG_SZ	Symantec Endpoint Protection PRD 12.1.4013.4013
Identifier	REG_SZ	UNV_ENU_Symantec_Endpoint_Protection_PRD_12140134013_01
InstallDate	REG_SZ	201411210131
InstallState	REG_SZ	Installed
Language	REG_SZ	ENU
Manufacture	REG_SZ	Symantec
ProductName	REG_SZ	Endpoint Protection PRD
Version	REG_SZ	12.1.4013.4013
VersionShort	REG_SZ	12

After uninstalling the package, it’s change the install state to "Uninstalled", in addition is the uninstall Date Entered:

Name	Typ	Daten
(Standard)	REG_SZ	(Wert nicht festgelegt)
BuildNumber	REG_SZ	01
Company	REG_SZ	UNV
DisplayName	REG_SZ	Symantec Endpoint Protection PRD 12.1.4013.4013
Identifier	REG_SZ	UNV_ENU_Symantec_Endpoint_Protection_PRD_12140134013_01
InstallDate	REG_SZ	201411210131
InstallState	REG_SZ	Uninstalled
Language	REG_SZ	ENU
Manufacture	REG_SZ	Symantec
ProductName	REG_SZ	Endpoint Protection PRD
UninstallDate	REG_SZ	201501180216
Version	REG_SZ	12.1.4013.4013
VersionShort	REG_SZ	12

7.2.3 Reboot

A call to the function SF-SystemReboot, or if another Scripting Framework function registered a restart, the restart information will means flag in the registry cached (0 or 1):



At the end of the package installation is done, Scripting Framework checking if a reboot is pending. If a reboot was registered, Scripting Framework completed with return code 3010. The reboot is not performed by Scripting Framework itself! Registered by the return code, Microsoft SCCM will be automatically reboot and executed this by the System Center. The reboot flags are cached in the registry, that means that another distribution systems can be read the values, if necessary. In the rule the values are not needed. Most of the distribution system are in the position to evaluate the return code.

Caution: Each time you start the PowerShell script, the flags will be reset.

7.2.4 Variables

The default variables which are in a Scripting Framework script available, are written dynamically at the start of the framework. The step is performed only if the boot time of the client (computer variables) or the logon time (user variables) does not match. This query takes place due to performance optimizations.

In addition to the default variable it's possible to load (within the script) dynamically more variables, from a CFG file, with the function *SF-LoadVariables*. Individual variables can be set or removed with the command *SF-Set*. The variables are immediately available and are ready for use.

In the Scripting Framework the variables are adressed with *%VariableName%*. **Example:**

```
# Run
SF-Run "%_Windows%\notepad.exe" "" -Show
```

The command starts the Notepad, the variable %_Windows% is dissolved by Scripting Framework → C:\Windows\Notepad.exe

7.2.4.1 Machine - Default variables

Name:	Value:
_CommonAdminTools	C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Administrative Tools
_CommonApplicationData	C:\ProgramData
_CommonDesktop	C:\Users\Public\Desktop
_CommonDocuments	C:\Users\Public\Documents
_CommonFiles	**32-Bit System:** C:\Program Files\Common Files **64-Bit System:** C:\Program Files (x86)\Common Files
_CommonFiles32	**32-Bit System:** C:\Program Files\Common Files **64-Bit System:** C:\Program Files (x86)\Common Files
_CommonFiles64	**32-Bit und 64 Bit System:** C:\Program Files\Common Files
_CommonMusic	C:\Users\Public\Music
_CommonPictures	C:\Users\Public\Pictures
_CommonPrograms	C:\ProgramData\Microsoft\Windows\Start Menu\Programs
_CommonStartMenu	C:\ProgramData\Microsoft\Windows\Start Menu
_CommonStartup	C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup
_CommonTemplates	C:\ProgramData\Microsoft\Windows\Templates
_CommonVideos	C:\Users\Public\Videos
_ComputerChassisName	Chassis Type Name. Example: Tablet
_ComputerChassisType	Chassis Type (Win32_Systemenclosure)
_ComputerDomain	Computer Domain oder Workgroup

_ComputerFQDN	COMPUTERNAME.domain.ch
_ComputerHardwareType	Physically oder Virtually
_ComputerHardwareTypeDetail	Hyper-V, Virtual PC, Xen, VMWare, None
_ComputerModel	Model Typ from the Computer. Example: HP Compaq Elite 8300 SFF
_ComputerModelVersion	Model Version from the Computer
_ComputerName	Name of the Computer
_ComputerOU	Domain OU des Computers
_ComputerType	WORKSTATION, NOTEBOOK, SERVER or CITRIX
_Fonts	C:\Windows\Fonts
_GroupAdministrators	Administratoren
_GroupEveryone	Everyone
_GroupGuests	Guests
_GroupPowerUsers	Mainuser
_GroupRemoteDesktopUsers	Remotedesktopuser
_GroupUsers	User
_OfficePlatform	Office Platform (x86 or x64)
_OSLang	1031
_OSType	x64
_OSVersion	10.0.19042
_OSVersionCurrent	10.0 (Major.Minor)
_ProgramFiles	**32-Bit System:** C:\Program Files **64-Bit System:** C:\Program Files (x86)
_ProgramFiles32	**32-Bit System:** C:\Program Files **64-Bit System:** C:\Program Files (x86)
_ProgramFiles64	**32-Bit und 64Bit System:** C:\Program Files
_Public	C:\Users\Public
_sProgramFiles	C:\PROGRA~2
_sProgramFiles32	C:\PROGRA~2
_sProgramFiles64	C:\PROGRA~1
_SystemDrive	C:
_Temp	C:\Windows\TEMP
_UserAdministrator	Administrator
_UserSystem	NT AUTORITÄT\SYSTEM
_Windows	C:\Windows
_WindowsSystem	**32-Bit System:** C:\Windows\system32 **64-Bit System:**

	C:\Windows\SysWOW64
_WindowsSystem32	**32-Bit System:** C:\Windows\system32 **64-Bit System:** C:\Windows\SysWOW64
_WindowsSystem64	**32-Bit und 64-Bit System:** C:\Windows\system32

7.2.4.2 Machine - Runtime variables

Name:	Value:
_AppVPackageRoot	Within an App-V packages is this variable available, which shows in the App-V root
_PkgBuildNumber	01
_PkgCache	C:\Windows_ScriptingFramework\Cache\PkgIdentifier (look InstallUser.ps1)
_PkgCFGName	PkgIdentifier (from XML)
_PkgDisplayName	Manufacturer ProductName Version (from XML)
_PkgIdentifier	PkgIdentifier (from XML)
_PkgIdentifierShort	PkgIdentifier without Version and BuildNumber (from XML)
_PkgLang	Language Code (generated with the function SF-Language)
_PkgLanguage	Language (from XML)
_PkgManufacturer	Manufacturer (from XML)
_PkgProductName	ProductName (from XML)
_PkgSource	Path from which the package was started
_PkgVersion	Detailed Version (z.B. 14.0.0.179)
_PkgVersionShort	Short Version (from XML)

7.2.4.3 User - Default variables

These variables are in the install user.ps1 or in the scripting mode available:

Name:	Value:
_ApplicationData	C:\Users*Username*\AppData\Roaming
_Cookies	C:\Users*Username*\AppData\Roaming\Microsoft\Windows\Cookies
_Desktop	C:\Users*Username*\Desktop
_Favorites	C:\Users*Username*\Desktop
_History	C:\Users*Username*\AppData\Local\Microsoft\Windows\History
_HomeDrive	H:
_HomePath	\Users*Username*
_HomeShare	
_InternetCache	C:\Users*Username*\AppData\Local\Microsoft\Windows\Temporary Internet Files
_LastLogonTime	12/17/2014 07:48:51
_LocalApplicationData	C:\Users*Username*\AppData\Local

_LocalLowApplicationData	C:\Users*Username*\AppData\LocalLow
_MyDocuments	C:\Users*Username*\Documents
_MyMusic	C:\Users*Username*\Music
_MyPictures	C:\Users*Username*\Pictures
_MyVideos	C:\Users*Username*\Videos
_Personal	C:\Users*Username*\Documents
_Programs	C:\Users*Username*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs
_Recent	C:\Users*Username*\AppData\Roaming\Microsoft\Windows\Recent
_SendTo	C:\Users*Username*\AppData\Roaming\Microsoft\Windows\SendTo
_StartMenu	C:\Users*Username*\AppData\Roaming\Microsoft\Windows\Start Menu
_Startup	C:\Users*Username*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
_Temp	C:\Users*Username*\AppData\Local\Temp
_Templates	C:\Users*Username*\AppData\Roaming\Microsoft\Windows\Templates
_User	*Username*
_UserProfile	C:\Users*Username*

7.2.4.4 User - Default variables (Active Directory)

Caution: This information can only be read if the AD is present or accessible.

Name:	Value:
_City	Bischofszell
_Company	Windows Client Management AG
_Country	CH
_Department	Informatik
_Description	Domain Admin ohne GPO und Roaming Profile
_DisplayName	Test1
_EMail	test1@wincm.ch
_Fax	071 451 14 41
_FirstName	Test1
_Homepage	www.wincm.ch
_Initials	TT
_LastName	Test
_Office	Office 101
_Pager	10 72
_PostalCode	9220
_Province	TG
_Street	Industriestrasse 6
_Telephone	n/A
_TelephoneHome	

_TelephoneIP	
_TelephoneMobile	
_Title	Microsoft Certified Enterprise Administrator

7.2.4.5 User - Runtime variables

Name:	Value:
_AppVPackageRoot	Within an App-V packages is this variable available, which shows in the App-V root
_PkgBuildNumber	01
_PkgCache	C:\Windows_ScriptingFramework\Cache\PkgIdentifier (siehe InstallUser.ps1)
_PkgCFGName	PkgIdentifier (from XML)
_PkgDisplayName	Manufacturer ProductName Version (from XML)
_PkgIdentifier	PkgIdentifier (from XML)
_PkgIdentifierShort	PkgIdentifier without Version and BuildNumber (from XML)
_PkgLang	Language Code (generated with the function SF-Language)
_PkgLanguage	Language (from XML)
_PkgManufacturer	Manufacturer (from XML)
_PkgProductName	ProductName (from XML)
_PkgSource	Path from which the package was started
_PkgVersion	Detailed Version (z.B. 14.0.0.179)
_PkgVersionShort	Short Version (from XML)

7.3 Command Line Options (Install.exe and Uninstall.exe)

The Install.exe and Uninstall.exe support the following optional parameters:

Name:	Description:
-AdditionalLog	You can use this parameter to create your own log for each package. **Example:** Install.exe -AdditionalLog "C:\Logs\Example.log"
-DisableCustomFinalize	Disables the Custom Finalize phase.
-DisableCustomInitialize	Disables the Custom Initialize phase.
-DisableLogging	The log output is deactivated. The parameter can be combined with the Switch -AdditionalLog: **Example:** Install.exe -AdditionalLog "C:\Logs\Example.log" -DisableLogging
-NoError	At a error (1603) still the return code of 0 is returned.
-NoReboot	Instead of a 3010 return code 0 will be returned.
-PerformanceMode	Scripting Framework starts with the best possible performance to run the script as fast as possible. For this, some checks and steps are omitted. The mode should only be used in scripting mode.
-Silent	No display of the installation process.

7.4 Logs and Errorcodes

Each function of Scripting Framework writes detailed information to the log, whereby each step is understandable. There are return standard return codes, which can be analyzed independently of the software deployment solution.

7.4.1 Log files

Computer Log „C:\Windows_ScriptingFramework\Logs“ and User Log (%LOCALAPPDATA%_ScriptingFramework\Logs) between information, warnings and errors are distinguished. Over a longer period of time to understand the events in the log, the log files are archived according to 1 MB, without affecting the performance of the script to influence.

Log Filename:	Description:
._ScriptingFramework\Logs\ ScriptingFramework.log	This file contains the current entries.
._ScriptingFramework\Logs\ScriptingFramework_1.log	Archivenumber 1
._ScriptingFramework\Logs\ScriptingFramework_2.log	Archivenumber 2
._ScriptingFramework\Logs\ScriptingFramework_3.log	Archivenumber 3
._ScriptingFramework\Logs\ScriptingFramework_4.log	Archivenumber 4
._ScriptingFramework\Logs\ScriptingFramework_5.log	Archivenumber 5 (oldest Entries)

Example ScriptingFramework.log:

```
17:01:2015-21:25:34   INFO: ***** © Copyright 2013 Windows Client Management AG - www.wincm.ch **************************
17:01:2015-21:25:35   INFO: ***** ScriptingFramework.exe ************************************************************
17:01:2015-21:25:35   INFO:      Command Line: "C:\WINDOWS\_ScriptingFramework\Modul\ScriptingFramework.exe"
"\\srvdeployment01\PkgSources\ UNV_MUI_Adobe_Photoshop_Elements_11_01\Classic\\Install.ps1" "0000"
17:01:2015-21:25:35   INFO:      Install Script: \\srvdeployment01\PkgSources\
UNV_MUI_Adobe_Photoshop_Elements_11_01\Classic\Install.ps1
17:01:2015-21:25:35   INFO:      LanguageCode: 0000
17:01:2015-21:25:35   INFO: ***** START Initialization Version 1.6.3.0 ******************************************************
```

```
17:01:2015-21:25:35   INFO: ===== Get Machine Variables ================================================================
17:01:2015-21:25:37   INFO:      Script is not running in InstallUser mode, user actions are not performed
17:01:2015-21:25:37   INFO: ===== ResetRebootFlag ======================================================================
17:01:2015-21:25:37   INFO: ***** END Initialization
*****************************************************************************************************************
17:01:2015-21:25:37   INFO: ***** START Installation
*****************************************************************************************************************
17:01:2015-21:25:37   INFO: ===== GetPackageDefinition ==================================================================
17:01:2015-21:25:37   INFO:      File:        \\srvdeployment01\PkgSources\ UNV_MUI_Adobe_Photoshop_Elements_11_01\Classic
\Package.xml
17:01:2015-21:25:37   INFO:      DisplayName:   Adobe Photoshop Elements 11 MUI
17:01:2015-21:25:37   INFO:      Identifier:    UNV_MUI_Adobe_Photoshop_Elements_11_01
17:01:2015-21:25:37   INFO:      Manufacturer:  Adobe
17:01:2015-21:25:37   INFO:      ProductName:   Photoshop Elements
17:01:2015-21:25:37   INFO:      Version:      11
17:01:2015-21:25:37   INFO:      VersionShort:  11
17:01:2015-21:25:37   INFO:      Language:     MUI
17:01:2015-21:25:37   INFO:      BuildNumber:   01
17:01:2015-21:25:37   INFO:      Company:      UNV
17:01:2015-21:25:37   INFO: ===== Taskkill ============================================================================
17:01:2015-21:25:37   INFO:      ProcessOrFolder: C:\Program Files (x86)\Adobe
17:01:2015-21:25:37   INFO:      Stop-Process: C:\Program Files (x86)\Adobe
17:01:2015-21:25:37   INFO: ===== Taskkill ============================================================================
17:01:2015-21:25:37   INFO:      ProcessOrFolder: C:\Program Files (x86)\Adobe Media Player
17:01:2015-21:25:37   INFO:      Stop-Process: C:\Program Files (x86)\Adobe Media Player
17:01:2015-21:25:37   INFO: ===== Taskkill ============================================================================
17:01:2015-21:25:37   INFO:      ProcessOrFolder: C:\Program Files (x86)\Common Files\Adobe
17:01:2015-21:25:37   INFO:      Stop-Process: C:\Program Files (x86)\Common Files\Adobe
17:01:2015-21:25:37   INFO: ===== LoadVariables ========================================================================
17:01:2015-21:25:38   INFO:      Folder Level 1:
\\pkgconfig\Config_Eng\Pkg_Config_Files\MGB\UNV_MUI_Adobe_Photoshop_Elements_11_01.cfg
17:01:2015-21:25:38   INFO:      Folder Level 2:
\\pkgconfig\Config_Eng\Pkg_Config_Files\UNV_MUI_Adobe_Photoshop_Elements_11_01.cfg
17:01:2015-21:25:38   INFO:      INISection: CommonClientSettings
17:01:2015-21:25:38   INFO:      License = 1057-4754-0084-0281-4585-8195
17:01:2015-21:25:38   INFO: ===== Language ============================================================================
17:01:2015-21:25:38   INFO:      1031: German
17:01:2015-21:25:38   INFO:      1033: English
17:01:2015-21:25:38   INFO:      1036: French
17:01:2015-21:25:38   INFO:      1040: Italian
17:01:2015-21:25:38   INFO:      ValidLanguages: 1031,1033,1036
17:01:2015-21:25:38   INFO:      Set _PkgLang (Detected OS Language): 1031
17:01:2015-21:25:38   INFO: ===== Installed ===========================================================================
17:01:2015-21:25:38   INFO:      ProductCode: {1D181764-DCD0-41B8-AA7B-0A599F027A72}
17:01:2015-21:25:38   INFO:      Result: False
17:01:2015-21:25:38   INFO: ===== Installed ===========================================================================
17:01:2015-21:25:38   INFO:      ProductCode: {D4D065E1-3ABF-41D0-B385-FC6F027F4D00}
17:01:2015-21:25:38   INFO:      Result: False
17:01:2015-21:25:38   INFO: ===== Run =================================================================================
17:01:2015-21:25:38   INFO:      Target: \\srvdeployment01\PkgSources\
UNV_MUI_Adobe_Photoshop_Elements_11_01\Classic\Setup\Setup.exe
```

```
17:01:2015-21:25:38   INFO:      Parameters: /UL1031 /V"SERIALNUMBER=1057-4754-0084-0281-4585-8195"
17:01:2015-21:25:38   INFO:      Timeout: 9999 (Seconds)
17:01:2015-21:27:08   INFO:      Reporting task Exitcode: (0)
17:01:2015-21:27:08   INFO: ===== Delete ================================================================================
17:01:2015-21:27:08   INFO:      File: C:\Users\Public\Desktop\Adobe Photoshop Elements 11.lnk
17:01:2015-21:27:08   INFO: ===== NTFSPerm ==============================================================================
17:01:2015-21:27:08   INFO:      FileOrFolder: C:\ProgramData\Adobe\Elements Organizer
17:01:2015-21:27:08   INFO:      User: Users
17:01:2015-21:27:08   INFO:      Permission: Modify
17:01:2015-21:27:08   INFO:      Command: icacls.exe "C:\ProgramData\Adobe\Elements Organizer" /grant "Users":(OI)(CI)M /Q /T /C
17:01:2015-21:27:08   INFO: ===== NTFSPerm ==============================================================================
17:01:2015-21:27:08   INFO:      FileOrFolder: C:\ProgramData\Adobe\Photoshop Elements
17:01:2015-21:27:08   INFO:      User: Users
17:01:2015-21:27:08   INFO:      Permission: Modify
17:01:2015-21:27:08   INFO:      Command: icacls.exe "C:\ProgramData\Adobe\Photoshop Elements" /grant "Users":(OI)(CI)M /Q /T /C
17:01:2015-21:27:10   INFO: ===== INIWrite ==============================================================================
17:01:2015-21:27:10   INFO:      INIFile: C:\ProgramData\Adobe\Elements Organizer\11.0\Elements.ini
17:01:2015-21:27:10   INFO:      Section: PSEInformation
17:01:2015-21:27:10   INFO:      Name: Country
17:01:2015-21:27:10   INFO:      Value: 223
17:01:2015-21:27:10   INFO: ***** Finalize
********************************************************************************************************************
17:01:2015-21:27:10   INFO:      PackageName: Adobe Photoshop Elements 11 MUI
17:01:2015-21:27:10   INFO:      User settings disabled: Disabled_InstallUser.ps1
17:01:2015-21:27:10   INFO:      System Reboot: No reboot is pending. Exit with Code: 0
17:01:2015-21:27:10   INFO: ***** END Installation
********************************************************************************************************************
```

7.4.2 Return values

Scripting Framework ended each with the following return codes, which are MSI compliant:

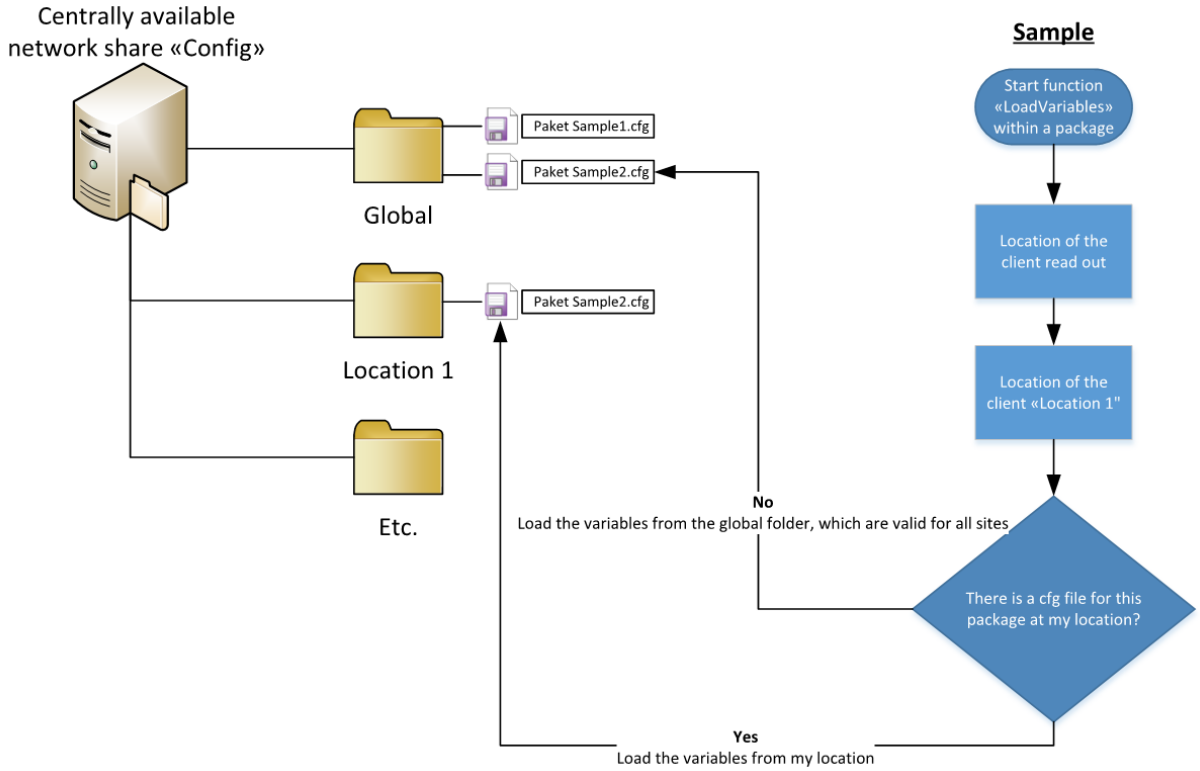
ErrorCode:	Description:
0	The action has been completed successfully.
1603	Fatal error during installation.
3010	To finish the installation, a reboot is required.
1641	Immediate Reboot

The error code is also visible in the Scripting Framework log.

7.5 Loading of package variables for dynamic packages

When loading variables with the function SF-LoadVariables, the system checks automatically if a site-specific .cfg file is available. The function c_ConfigPath of the transferred path, is automatically extended with the respective location. For the query of the location is the internal variable c_configcompany.

Process:



If no site-specific settings (Level 1) is available, the variables from the "global" valid folder (Level 2) are loaded, this corresponds to the value %c_ConfigPath%. In the global settings it is a configuration which are valid for all, regardless of location. Should also no configuration found under the level 2, an error is output. Example:

Scripting Framework Command:

```
SF-LoadVariables "%c_ConfigPath%\%_PkgCFGName%.cfg" "CommonClientSettings"
```

Registry Values (HKEY_LOCAL_MACHINE\SOFTWARE*Wow6432Node*\ScriptingFramework\Variables):
c_configpath = \\WINCM-SRV01.wincm.local\Config
c_configcompany = WinCM

Log Output:

```
29:04:2015-14:22:20   INFO: ===== LoadVariables ================================================================================
29:04:2015-14:22:21   INFO:      Folder Level 1: \\WINCM-SRV01.wincm.local\Config\WinCM\UNV_UNI_WinCM_Sample_10_01.cfg
29:04:2015-14:22:21   INFO:      Folder Level 2: \\WINCM-SRV01.wincm.local\Config\UNV_UNI_WinCM_Sample_10_01.cfg
```

7.6 Handling of user settings

The InstallUser.ps1 is used for setting user preferences. All functions of Scripting Framework is available and can be used as long the action does not require administrator rights.

Scripting Framework recognize during the installation of a package automatically if the user configuration is activated (InstallUser.ps1) or disabled (Disabled_InstallUser.ps1). If this is enabled, the entire configuration of the user folder is cached to "C:\Windows_ScriptingFramework\Cache*PackageIdentifier*" and the information is written accordingly in the registry. The name of the unique package identifier is generated from the data in Package.xml, where all points are removed ".". Uninstalling a package automatically removes the cache and registry entries.

If the configuration *DisableImmediateUserInstallation*=1 on the computer is not active, the user settings installing (for all logged on users on the device) after the installation automatically, applied by Scripting Framework. It is not necessary that the user needs to log off! Should a user not be logged on or the immediate installation has been disabled in the configuration, the user configuration is performed at the next login. See also under immediate user installation.

7.6.1 Technical description

When logging in a user, the Scripting Framework ActiveSetup.exe compares the two registry keys and their subkeys:

- HKEY_LOCAL_MACHINE\SOFTWARE*WOW6432Node*\ScriptingFramework\UserSettings
- HKEY_CURRENT_USER\SOFTWARE\ScriptingFramework\UserSettings

Below this path is a separate key for each package with user settings. Each of these subkeys contains the following values:

Registry Value:	Description:
(default)	The default value can contain any text, such as the program name or a description.
StubPath	The command line to be executed when the user logs on: "C:\Windows_ScriptingFramework\Modul\ScriptingFramework.exe" -File "C:\Windows_ScriptingFramework\Cache*PkgIdentifier*\User\InstallUser.ps1"
Version	The version is also compared. Is the version under HKLM greater than the version under HKCU then the StubPath runs again. Example: 1,0,0,1 Note: The version number is separated by a comma and not by points.

Important note: Since the Scripting Framework Version 2.5.0.0, the ActiveSetup from Microsoft is no longer used, but its own mechanism, which works analogously to MS ActiveSetup. The reason for this is that on a Windows 10 upgrade, all custom ActiveSetup keys are automatically removed from the registry. For the one-time migration to the internal user settings mechanism of Scripting Framework we offer a free script, please contact our support.

7.6.2 Procedure

1. A user logs in.
2. The ActiveSetup.exe checks the user settings under HKLM and compares them with the existing registry entries in the user profile HKCU.
3. If an entry is missing under HKCU or if the version in HKLM is greater than the version under HKCU, the command is executed in the registry value StubPath (from HKLM) and the entry is written to the registry (HKCU).

7.6.3 Published application (Citrix)

When a Published Application is called, the Explorer.exe process does not start, which also does not automatically call ActiveSetup.exe. This will not process any user settings from Scripting Framework. To solve this problem ActiveSetup.exe can be integrated into the login process as described in the MS article: http://support.microsoft.com/kb/195461/en-us

8 Deployment and configuration

8.1 Set up the central share (optional)

Set up first a share on a server, where later the CFG files for the dynamic packages can be stored (SF-LoadVariables). All devices that work with the ScriptingFramework, should have access to this share. Register this share with the configuration tool which we provide you.

8.2 Create a service account in AD (optional)

To access network drives (shares) requires the Scripting Framework a corresponding user. We recommend to create for this purpose a separate service account. The user needs only "Read" permissions for this share and no other rights are necessary. Enter the user also into the configuration tool which we provide you.

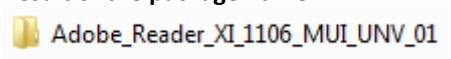
8.3 Defining the future package name (identifier)

Each package has a unique identifier, which is generated from the XML of the respective packet. In the configuration tool, you can define the name of the identifier:
PkgIdentifier = Manufacturer|ProductName|Version|Language|Company|BuildNumber

Example XML:

```
<Package>
        <BuildNumber>01</BuildNumber>
        <Company>UNV</Company>
        <Manufacturer>Adobe</Manufacturer>
        <ProductName>Reader XI</ProductName>
        <Version>11.0.6</Version>
        <VersionShort>11</VersionShort>
        <Language>MUI</Language>
</Package>
```

Result of the package name:

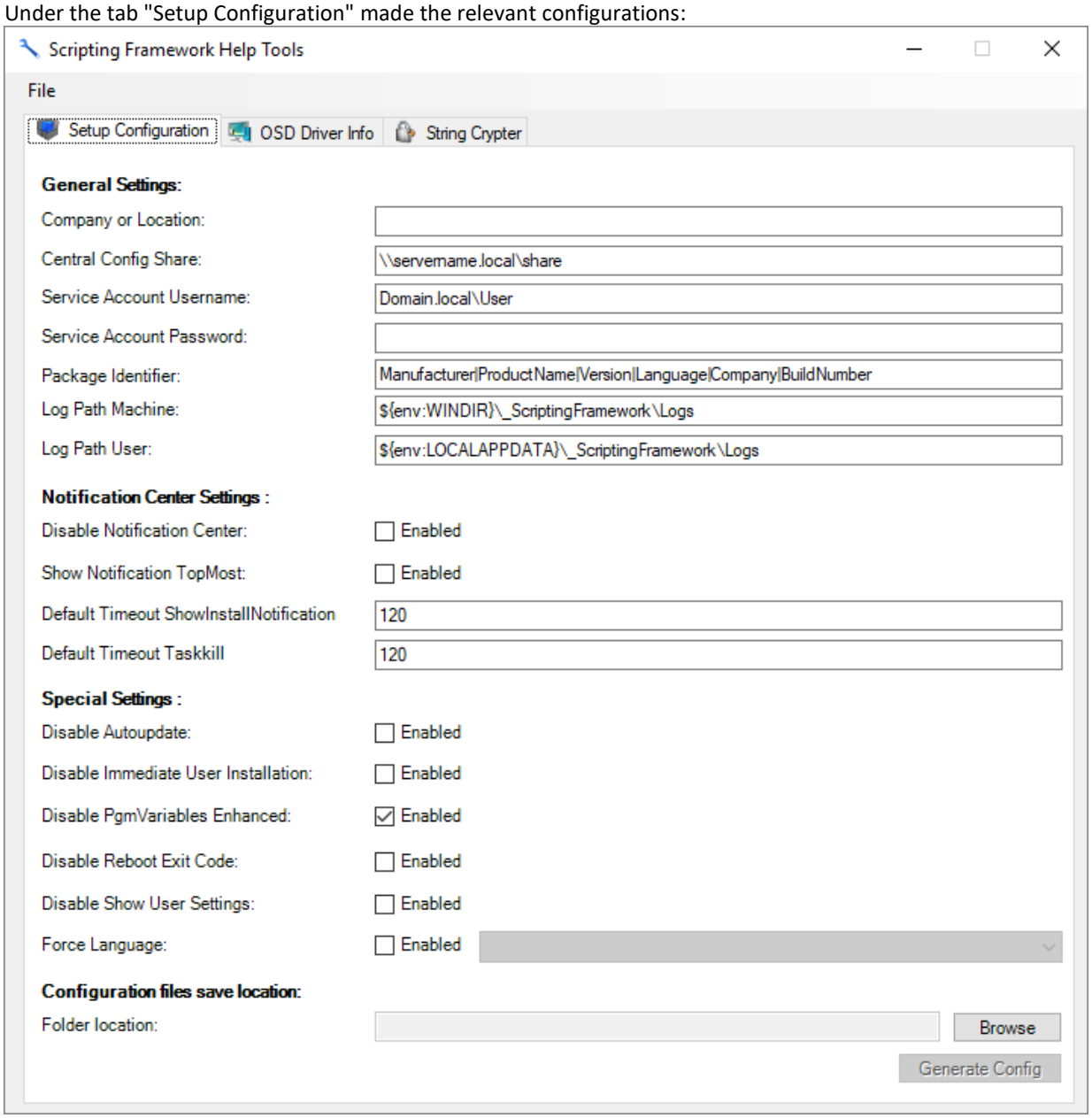


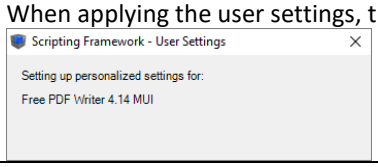
Caution: As soon packages used productively, the identifier can not be customized! More details about the structure of the XML file.

8.4 Perform initial configuration

Copy the installation package „WinCM_Scripting_Framework_1xxx_UNI_UNV_01“ to „C:\Temp“.
Now start the Help tools which are located in the newly copied installation package under the path „…\Classic\Setup\Tools\HelpTools.exe“.

Under the tab "Setup Configuration" made the relevant configurations:



Option	Description
General Settings	
Company or Location	Enter the company or the location. This value is used for the SF-LoadVariables function. For details, read the chapter Loading package variables for dynamic packages. This value will only be used in the beginning and will later be entered via SCCM task sequence or GPO with the correct location.
Central Config Share	Enter the recently created share (UNC path).
Service Account Username	Enter the created user who has “Read” permissions on the Config Share.
Service Account Password	Enter the user's password.
Package Identifier	Define the desired Packet Identifier. We recommend this to leave that way.
Log Path Machine	Path of the Scripting Framework Log files (Machine Actions)
Log Path User	Path of the Scripting Framework Log files (User Actions)
Notification Center Settings	
Disable Notification Center	Disables the Scripting Framework Notification Center. No messages will be display to the user, when using functions with notification option (eg SF-Taskkill).
Show Notification TopMost	The message of the Notification Center will be permanently displayed in the foreground.
Default Timeout ShowInstallNotification	The Notification Timeout (SF-ShowInstallNotification) in seconds, provided that no further displacement for the installation is possible. After this timeout (displays as a countdown for the user), the installation continues. With the value 0, the automatic time-out is disabled, but this is not the recommendation.
Default Timeout Taskkill	Timeout closed automatically the tasks in seconds (SF-Taskkill with Switch -Notification). With the value 0, the automatic time-out is disabled, but this is not the recommendation.
Special Settings	
Disable Autoupdate	Disables the Scripting Framework Autoupdate functionality, which is performed by the AutoUpdater.exe file.
Disable Immediate User Installation	The user settings (InstallUser.ps1) will automatically be applied after the package installation to all logged on users on the device. With this option, you can disable this feature.
Disable PgmVariables Enhanced	Disables the creation of the extended Pgm variables during a package installation.
Disable Reboot Exit Code	Unless Scripting Framework discovered a reboot, it returns a error code 0 instead of the return code 3010. This can, for example, be used on a Citrix server.
Disable Show User Settings	When applying the user settings, the window does not appear: 
Force Language	The function SF-Language which can be used in packages dynamically, reads each language and returned this. If it is desired to fix the language of SF-Language function, the language code can be entered here. In generally used only on language-specific Citrix servers.

Browse Folder	Enter the following path: C:\Temp\WinCM_Scripting_Framework_*1xxx*_UNI_UNV_01\Classic\Config

If all details were provided correctly, click generate config. Now check if the file "Config.p1 has been created correctly in the specified Config folder. The package for the engine installation is now finish created and can be used. Don’t forget to copy the file license.lic to "Classic\Setup\Modul_x64" and "Classic\Setup\Modul_x86".

8.5 Installation

Don’t forget to perform the initial configuration before installing Scripitng Framework. The prerequisites installations are based on a command line script, because without the corresponding PowerShell Version Scripting Framework will not work.

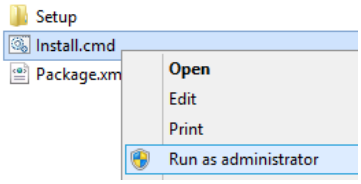
To execute the packages after the installation directly from the PowerShell ISE console, you must install the engineer as packetizer additional expansion, as described in the manual.

All required setup files and sample packages, can be found on our website: http://www.scriptingframework.com

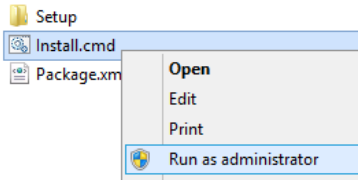
8.5.1 Manual installation

If the prerequisites are already installed you can skip these installations. Otherwise, use the setup routines from us to unpack the downloaded source code to C:\Temp.

Installation .Net Framework:
Run the file „C:\Temp\Microsoft_.Net_Framework_*4xx*_UNI_UNV_01\Classic\Install.cmd“ as administrator:

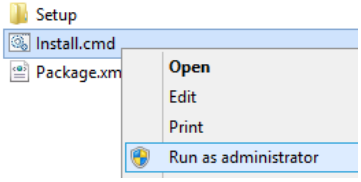


Installation PowerShell:
Run the file „C:\Temp\Microsoft_Powershell_*5x*_UNI_UNV_01\Classic\Install.cmd“ as administrator:



Caution: To ensure that the new PowerShell version can be used, a reboot is required.

Installation Scripting Framework
Run the file „C:\Temp\ WinCM_Scripting_Framework_*2xxx*_UNI_UNV_01\Classic\Install.cmd“ as administrator:



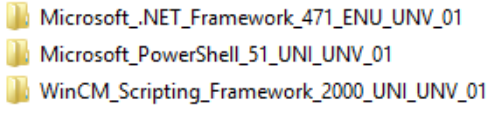
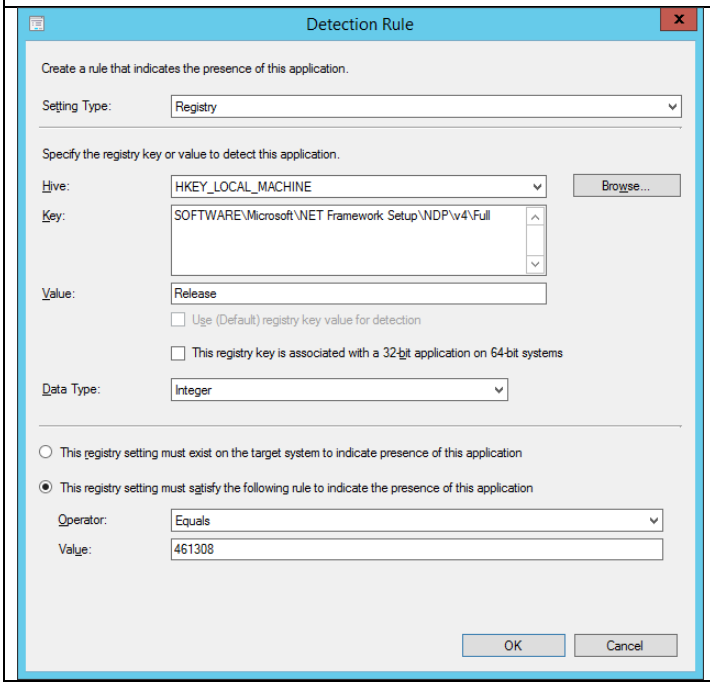
Caution: Make sure that you have previously used the "Initial Configuration" and the correct REG files are stored in the config folder.

8.5.2 Using software distribution (General)

To install Scripting Framework through your software distribution, create a corresponding package sleeve and start it with the supplied Install.cmd from us. There are no silent parameter, etc. necessary. Ensure that the prerequisites may also be installed.

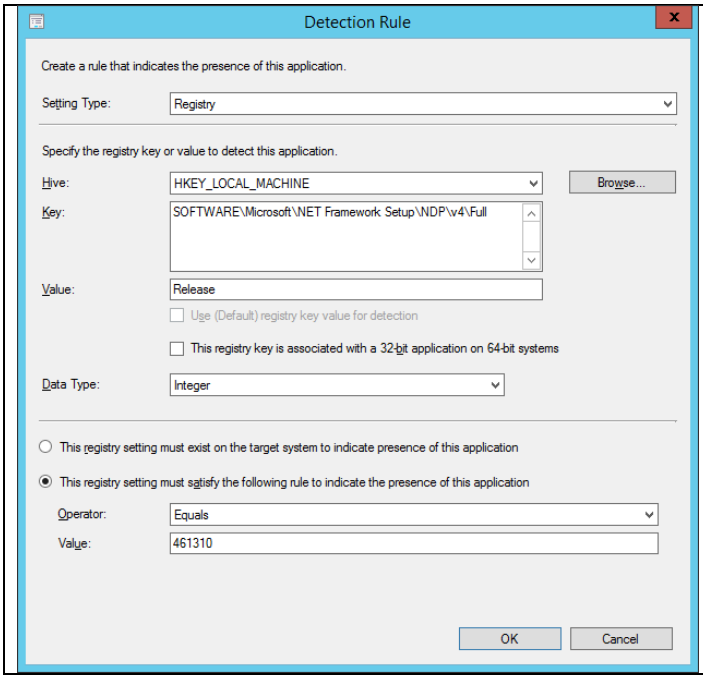
8.5.3 Using software distribution (Microsoft SCCM)

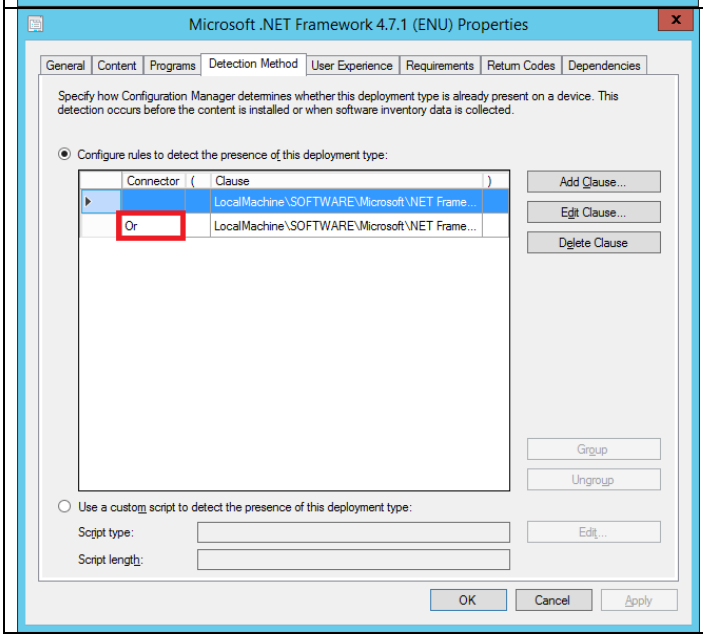
For the implementation of Microsoft System Center, the three packages must be recognized as an application. In the manual is only the creation of the Scripting Framework application full documented, step by step. For the prerequisites only the detection rule is specified.

<table>
<tr><td> 
</td><td>Copy the three downloaded packages to the SCCM Application Source.</td></tr>
<tr><td colspan="2">Microsoft .NET Framework</td></tr>
<tr><td> 
</td><td>Create first the Microsoft .NET Framework application.

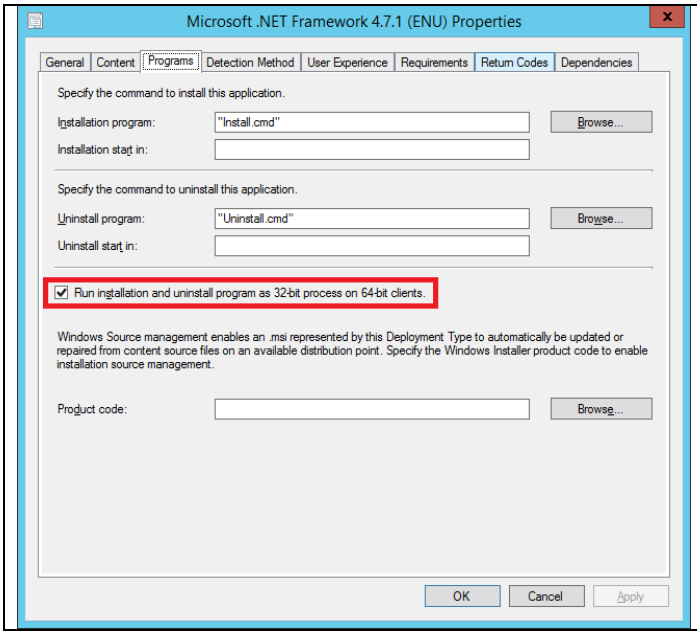
The detection rule can be found in the following three screenshots.

On Windows 10 Fall Creators Update: 461308
On all other OS versions: 461310</td></tr>
</table>



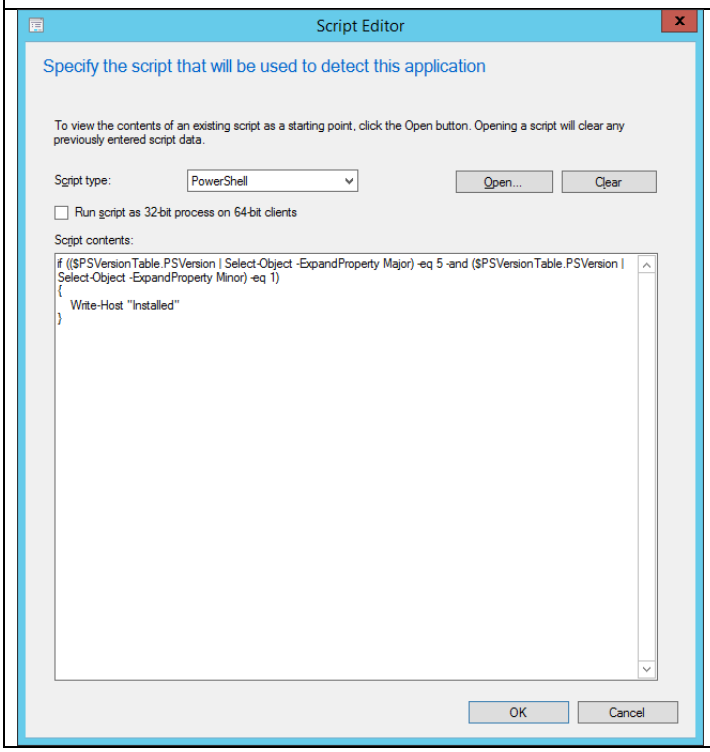


Use the Connector column to select the operator “Or”.



Important: For the .NET Framework installation you have to set the option "Run installation and uninstall program as 32-bit process on 64-bit clients". Otherwise an error occurs during the installation.

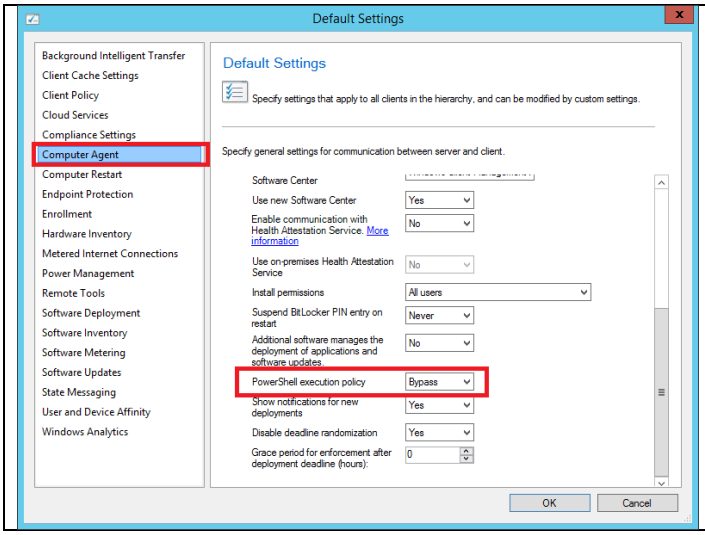
Microsoft PowerShell



Create next the PowerShell application with dependency on the .NET Framework package just created.

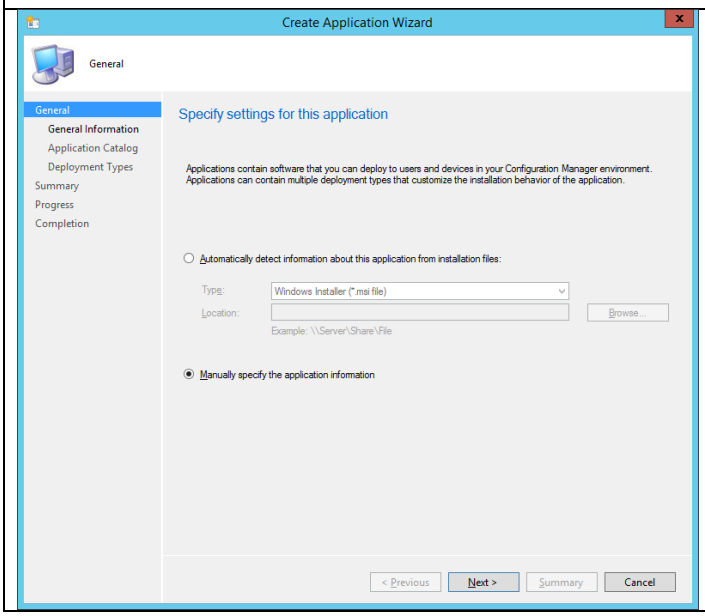
The detection rule used is the following PowerShell script:

```
if ((($PSVersionTable.PSVersion | Select-Object -ExpandProperty Major) -eq 5 -and ($PSVersionTable.PSVersion | Select-Object -ExpandProperty Minor) -eq 1)
{
   Write-Host "Installed"
}
```

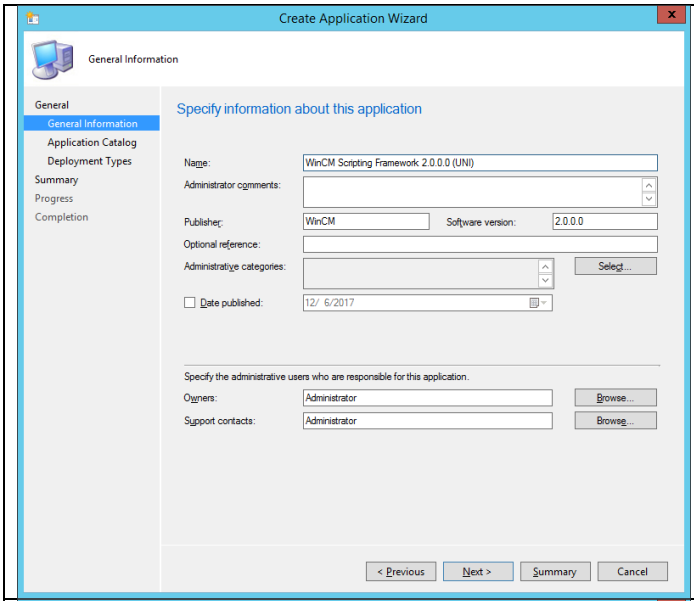
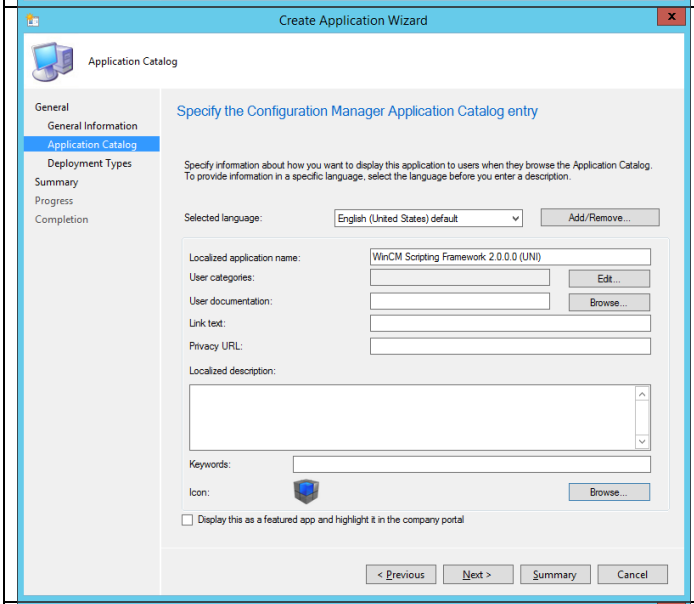
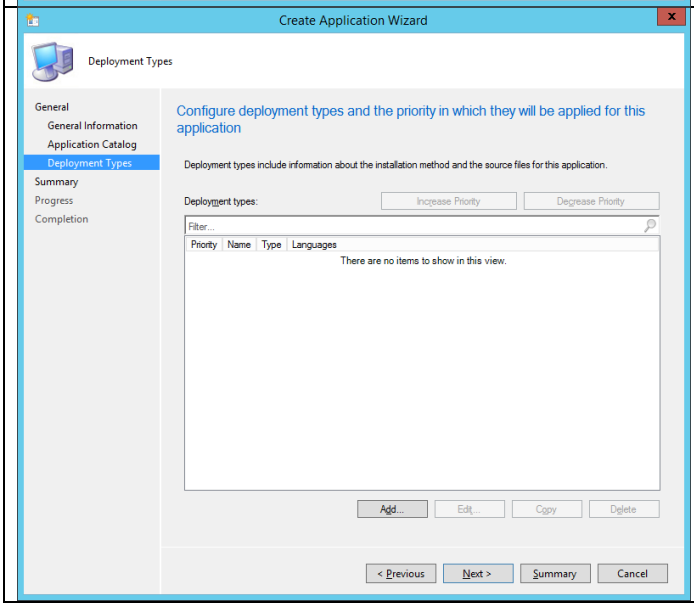


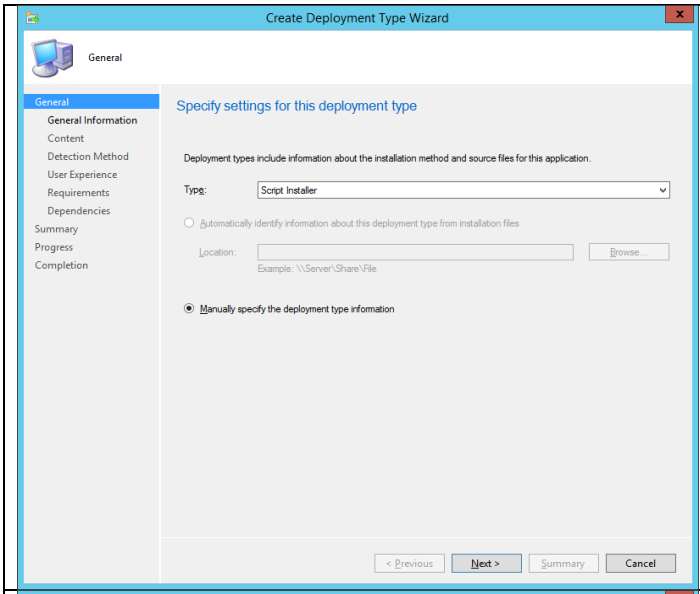
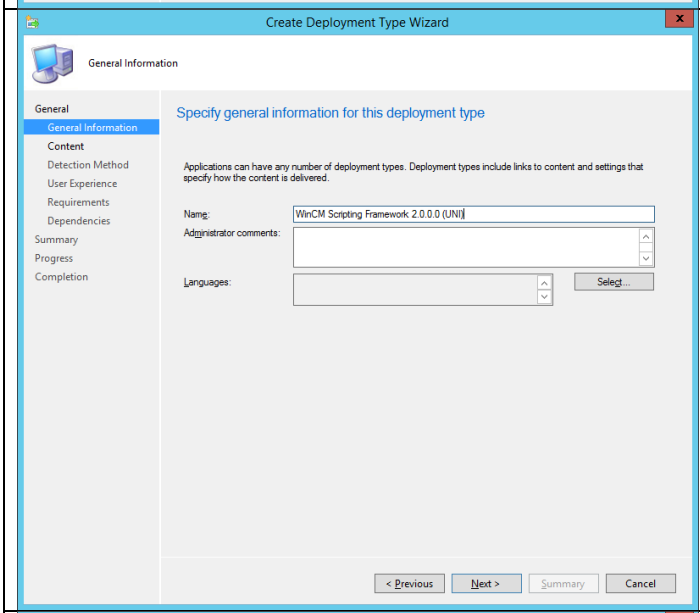
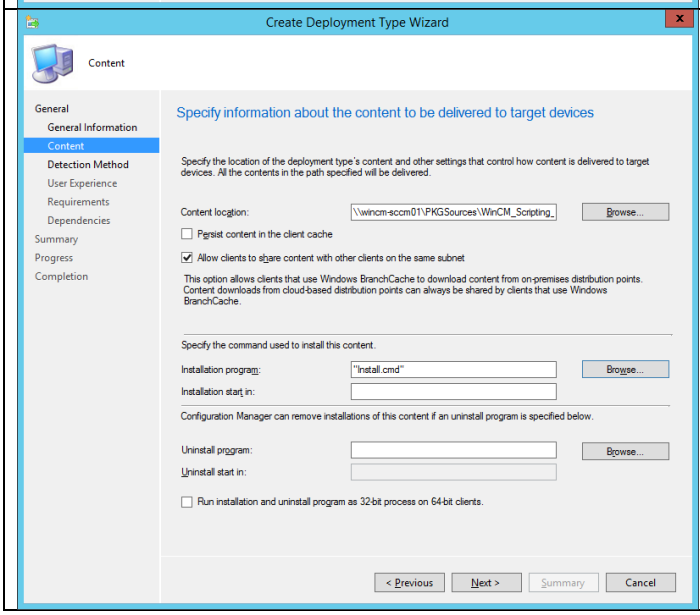
Important: For the PowerShell script detection rule, the PowerShell Execution Policy must be set to Bypass. See the screenshot.

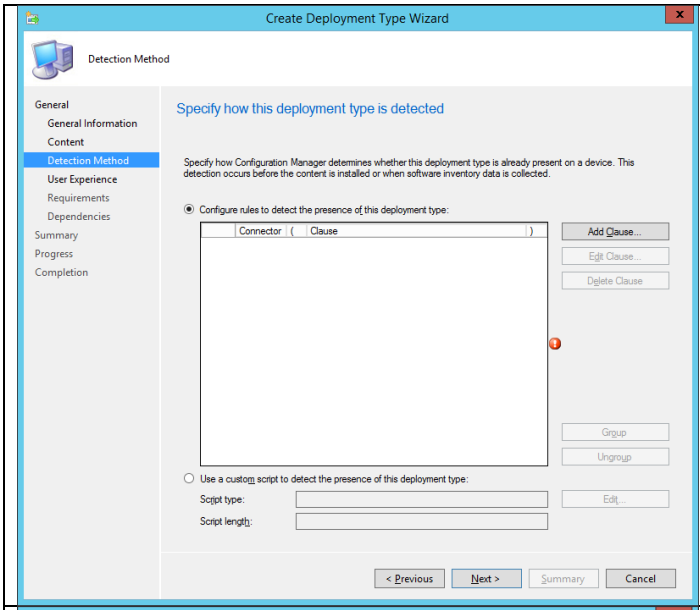
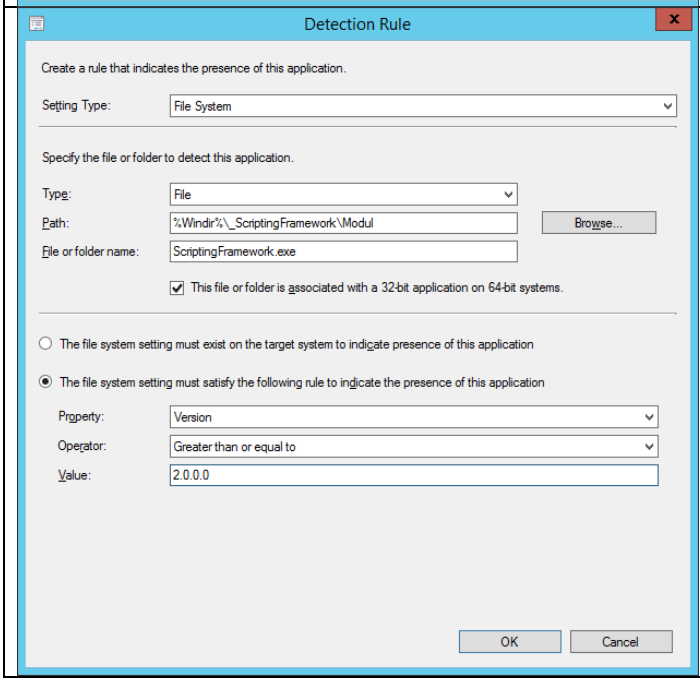
WinCM Scripting Framework (Step by step)

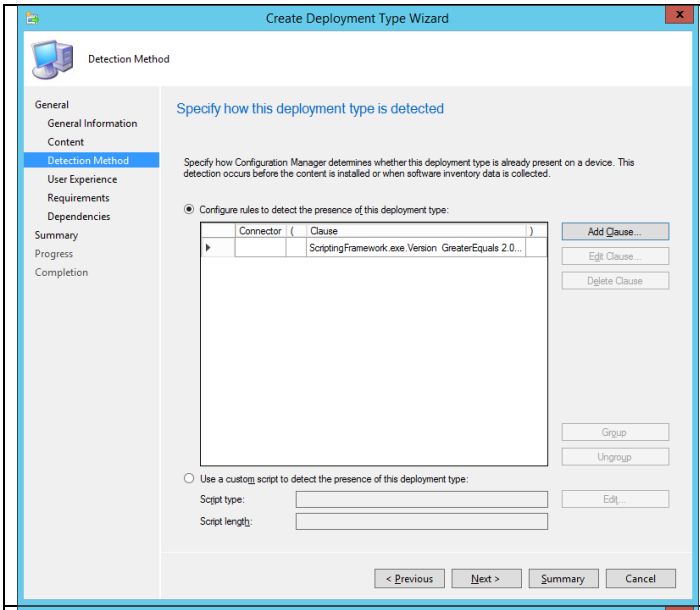
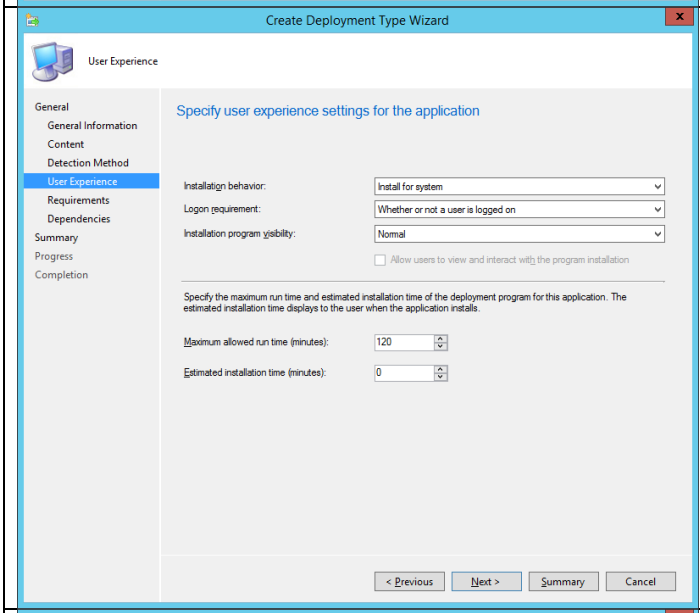
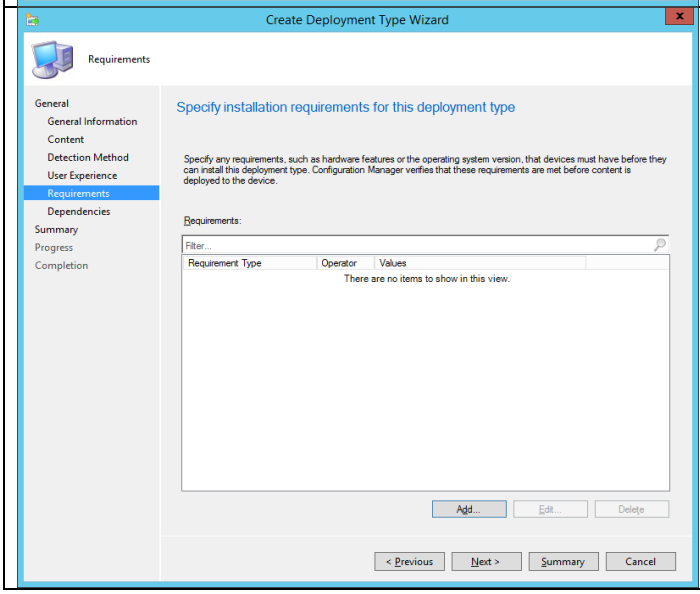


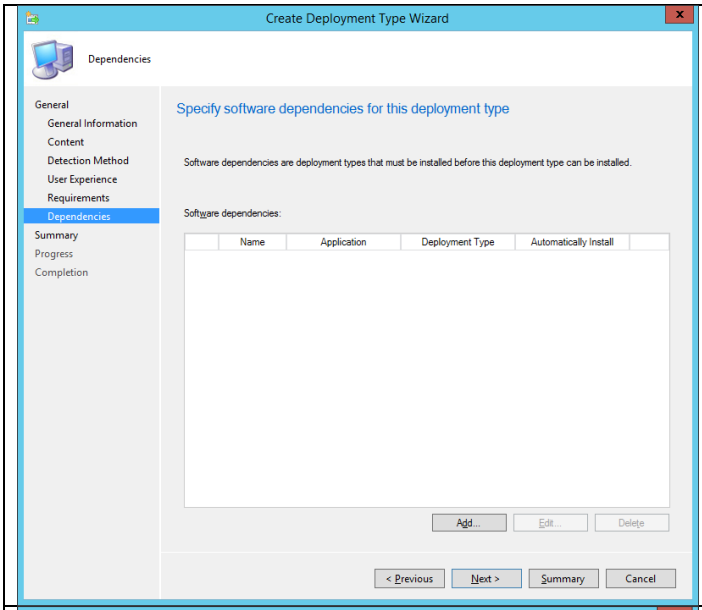
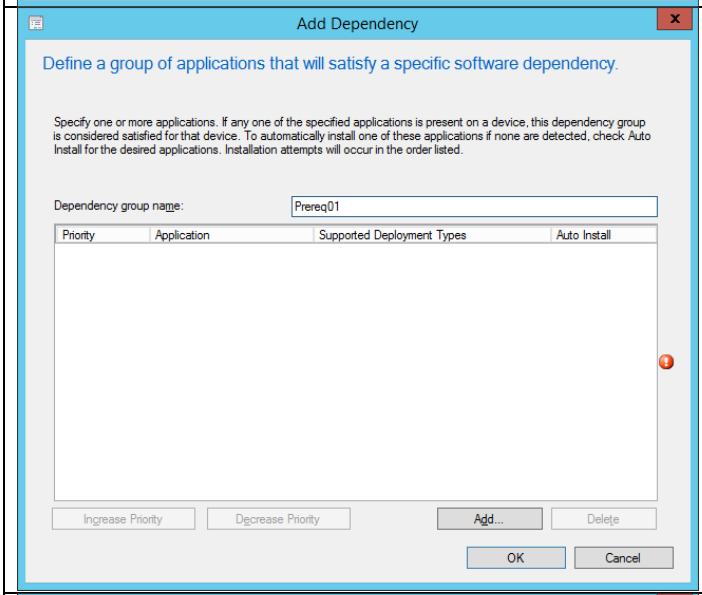
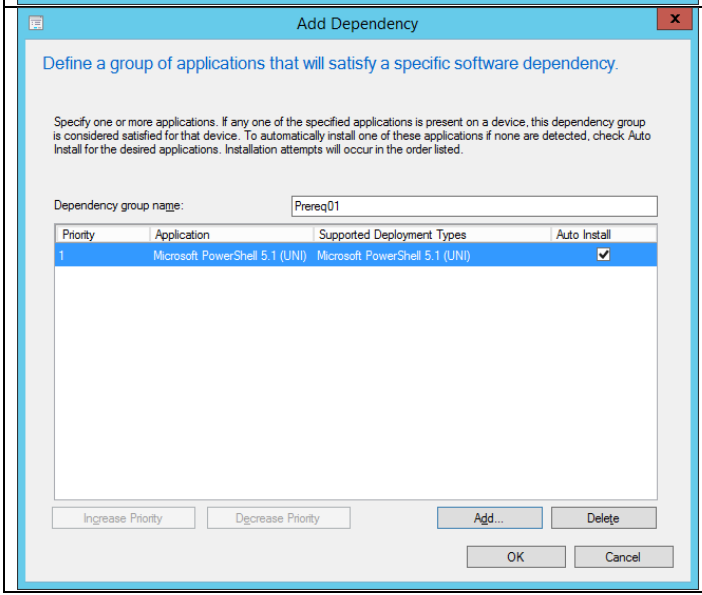
Create a new application.

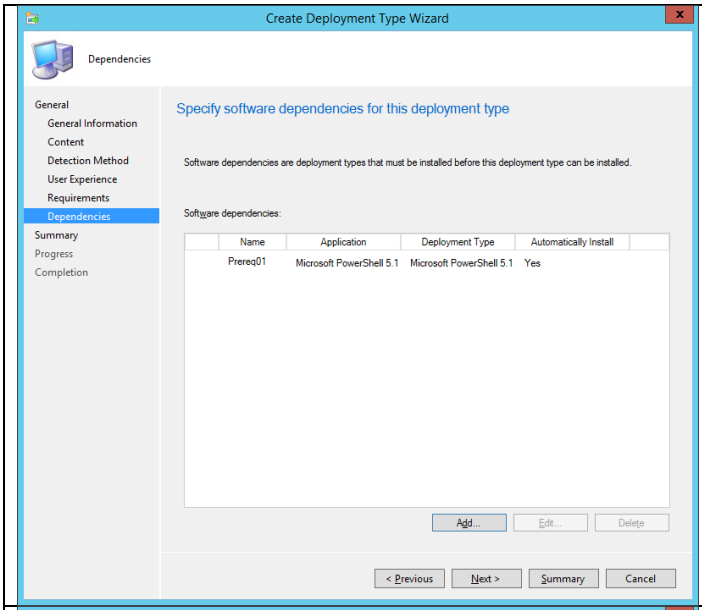
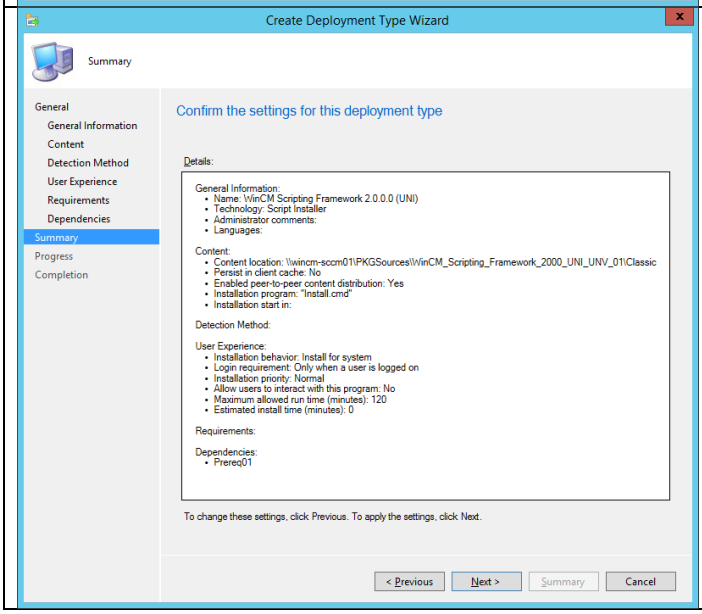
	Enter the relevant informations
	Enter the symbol. This can be found in the package in the folder Icon. Next
	Add

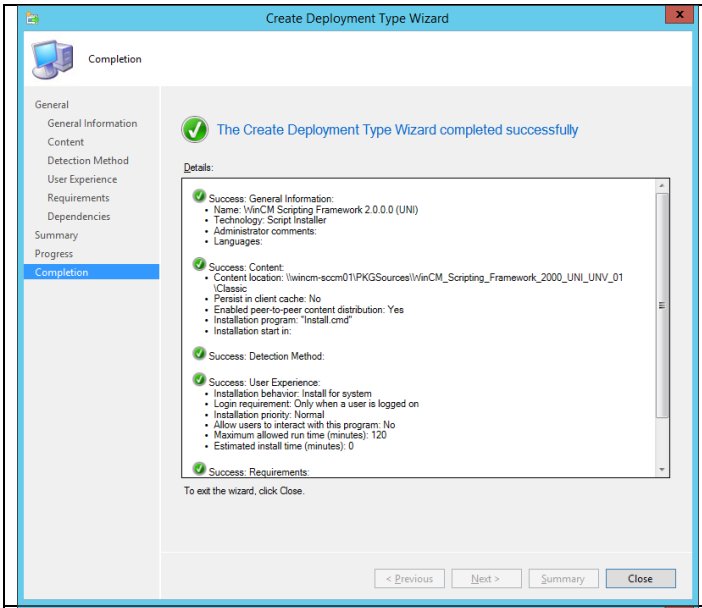
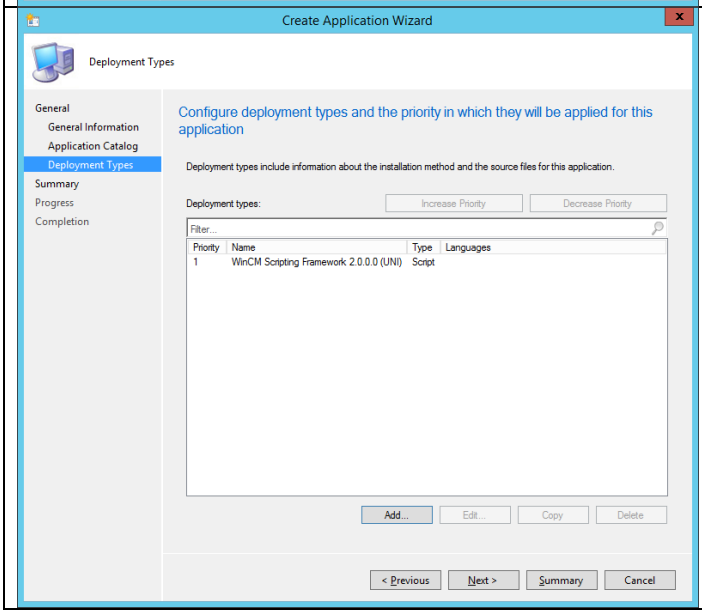
	**Type**: Script Installer
	Enter the name of the deployment. Next
	Content location: \\servername.local\share\PKGSources\WinCM_Scripting_Framework_2xxx_UNI_UNV_01\Classic Installation program: Install.cmd A remove is not provided, so there is no uninstall.

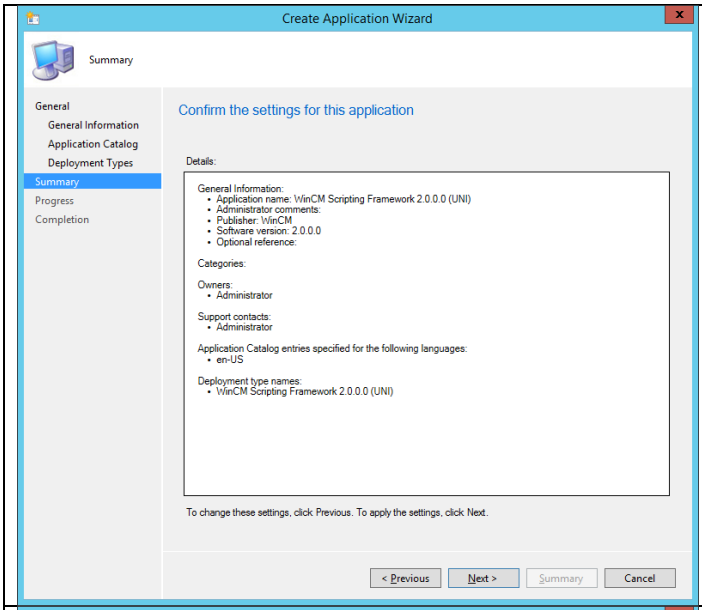
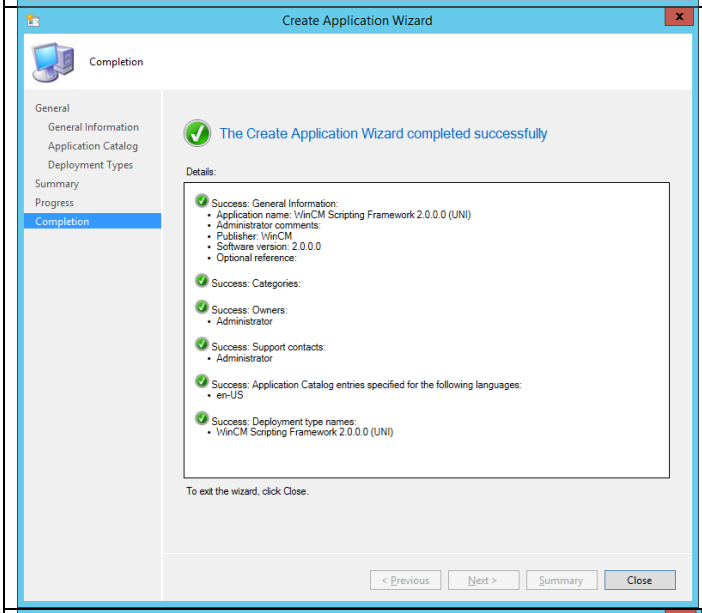
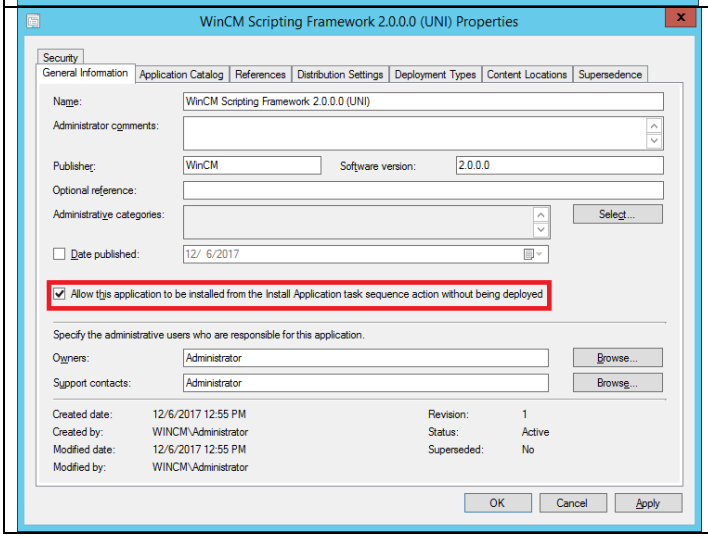
	Add Clause…
	Create a rule as shown on the screenshot. Adjust the value of the version of the Scripting Framework version used.

	Next
	**Installation behavior:** Install for system **Login requirement:** Wheter or not a user is logged on Next
	Next

	Add…
	Add as a dependency the previously acquired PowerShell application.
	OK

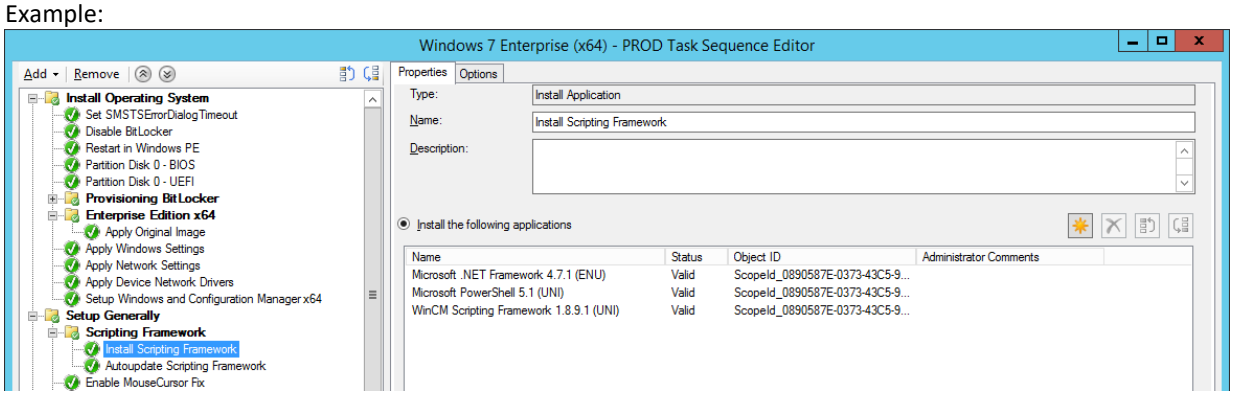
 Create Deployment Type Wizard Dependencies General General Information Content Detection Method User Experience Requirements Dependencies Summary Progress Completion Specify software dependencies for this deployment type Software dependencies are deployment types that must be installed before this deployment type can be installed. Software dependencies: Name Application Deployment Type Automatically Install Prereq01 Microsoft PowerShell 5.1 Microsoft PowerShell 5.1 Yes Add... Edit... Delete < Previous Next > Summary Cancel	Next
 Create Deployment Type Wizard Summary General General Information Content Detection Method User Experience Requirements Dependencies Summary Progress Completion Confirm the settings for this deployment type Details: General Information: • Name: WinCM Scripting Framework 2.0.0.0 (UNI) • Technology: Script Installer • Administrator comments: • Languages: Content: • Content location: \\wincm-sccm01\PKGSources\WinCM_Scripting_Framework_2000_UNI_UNV_01\Classic • Persist in client cache: No • Enabled peer-to-peer content distribution: Yes • Installation program: "Install.cmd" • Installation start in: Detection Method: User Experience: • Installation behavior: Install for system • Login requirement: Only when a user is logged on • Installation priority: Normal • Allow users to interact with this program: No • Maximum allowed run time (minutes): 120 • Estimated install time (minutes): 0 Requirements: Dependencies: • Prereq01 To change these settings, click Previous. To apply the settings, click Next. < Previous Next > Summary Cancel	Next

 Create Deployment Type Wizard Completion General General Information Content Detection Method User Experience Requirements Dependencies Summary Progress Completion The Create Deployment Type Wizard completed successfully Details: Success: General Information: Name: WinCM Scripting Framework 2.0.0.0 (UNI) Technology: Script Installer Administrator comments: Languages: Success: Content: Content location: \\wincm-sccm01\PKGSources\WinCM_Scripting_Framework_2000_UNI_UNV_01\Classic Persist in client cache: No Enabled peer-to-peer content distribution: Yes Installation program: "Install.cmd" Installation start in: Success: Detection Method: Success: User Experience: Installation behavior: Install for system Login requirement: Only when a user is logged on Installation priority: Normal Allow users to interact with this program: No Maximum allowed run time (minutes): 120 Estimated install time (minutes): 0 Success: Requirements: To exit the wizard, click Close. < Previous Next > Summary Close	
 Create Application Wizard Deployment Types General General Information Application Catalog Deployment Types Summary Progress Completion Configure deployment types and the priority in which they will be applied for this application Deployment types include information about the installation method and the source files for this application. Deployment types: Increase Priority Decrease Priority Filter... Priority Name Type Languages 1 WinCM Scripting Framework 2.0.0.0 (UNI) Script Add... Edit... Copy Delete < Previous Next > Summary Cancel	Next

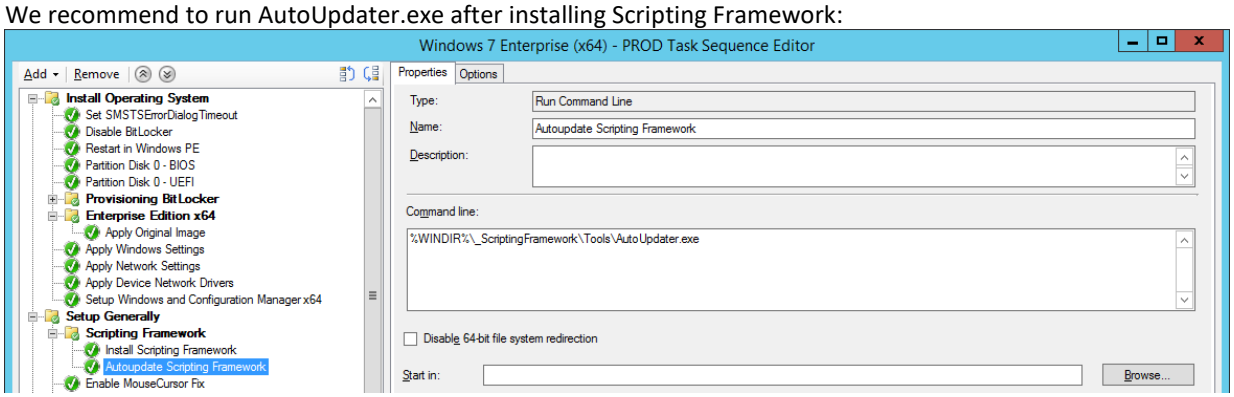
	Next
	Close
	Open the application properties and activate the option marked in the screenshot. You can now distribute the application to the distribution points. The application is now ready for deployment.

8.5.3.1 Task Sequence Integration

Scripting Framework must be installed as early as possible, before the first Scripting Framework package is used.
Example:



We recommend to run AutoUpdater.exe after installing Scripting Framework:

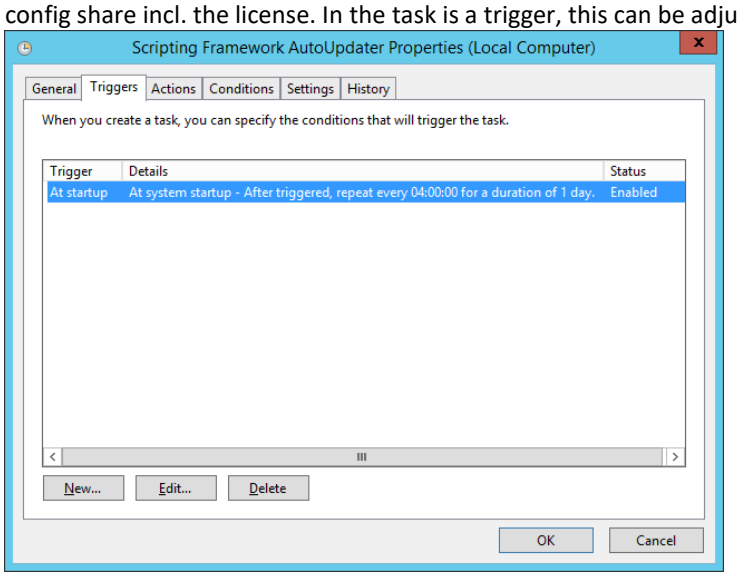


%WINDIR%_ScriptingFramework\Tools\AutoUpdater.exe

This is very useful, for example, when using a Capture Image on which the Detection method is already met. This step will always update the Scripting Framework version including the license.

8.6 Automatic update of Scripting Framework (AutoUpdate)

The setup of Scripting Framework (Install.cmd) created during the installation a schedule task named "Scripting Framework AutoUpdater". This makes it possible to update Scripting Framework files automatically via the config share incl. the license. In the task is a trigger, this can be adjusted if necessary, in the setup routine itself:



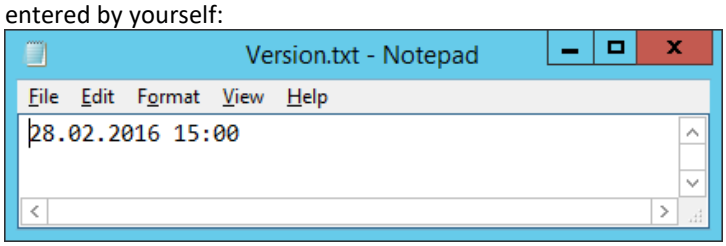
The Auto Update function requires very little resources and is usually done within seconds.

8.6.1 Operation

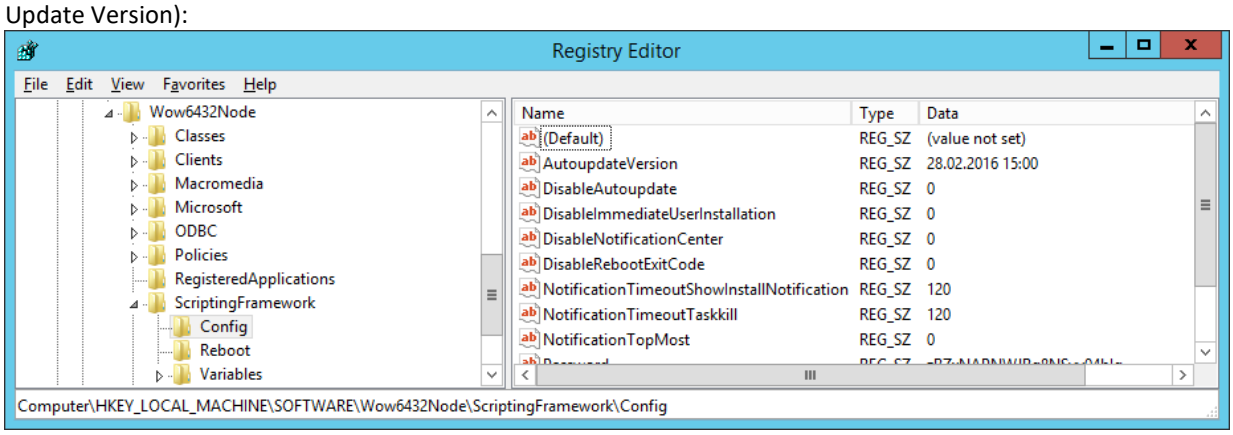
After run of the AutoUpdater.exe, this will be checked as the first step, if the functionality has been disabled on the value DisableAutoUpdate = 1 in the registry (config). If disabled, the AutoUpdater will terminate without performing another action.

The next step is to check whether the required information in the registry for the initial configuration are correct. This includes the values of user name, password and c_config path. On the basis of this is the authentication on the config share performed on which the Scripting Framework update files will be saved. If the "Scripting Framework AutoUpdate folder“ on the config share doesn’t exist or the defined configuration is not correct, the AutoUpdate automatically terminated.

Within the folder on the Share is the text file "Version.txt", in which the date and time of last modification is entered by yourself:



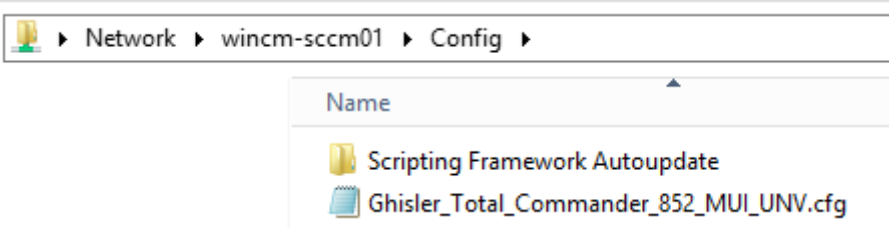
After a successful AutoUpdate the date from the text file will be deposited on the client in the registry (Auto Update Version):



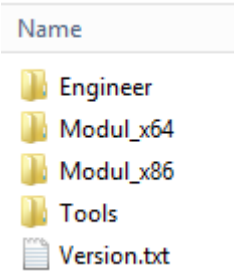
Only if these two informations do not match, the automatic update is performed respectively. The update of the license file takes place independently thereof. This is updated when the file date of the license on the share doesn't coincide with the local license file.

8.6.2 Required share structure

On the config share, the folder "Scripting Framework AutoUpdate" will be created. The entire structure can be found on our homepage for download:



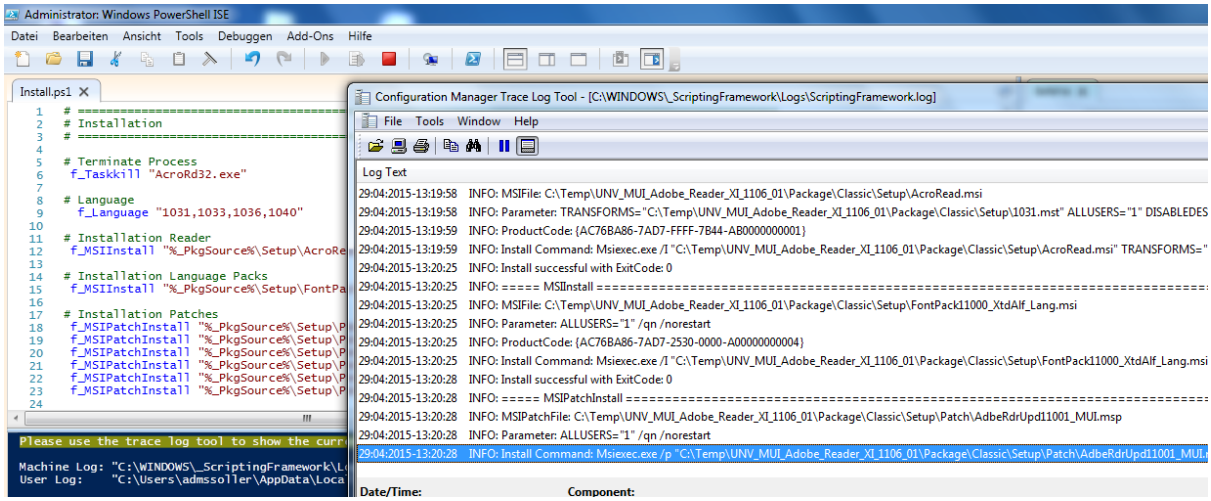
The structure of the Scripting Framework AutoUpdate folder:



Folder	Contents and Description
Engineer	The folder contains the engineer module "ScriptingFramework.psm1". This is updated on the client only, if it has been installed.
Modul_x64	All x64 module files including the personal license.
Modul_x86	All x86 module files including the personal license.
Tools	All Tools files of Scripting Framework.
Version.txt	The date and time of the last change are entered in this file (see operation).

8.7 Installation engineer extension (for the development of packages)

It exist a second setup routine to develop packages quickly and easily to start the packages directly from the PowerShell ISE console. This setup routine installed the appropriate extensions. Run the Install.cmd from the "Scripting Framework Engineer" as an administrator.

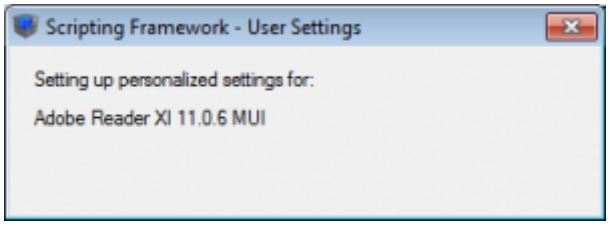


After the installation you can start the packages directly from the ISE console. We recommend to install the engineer expansion only on clients which creates Scripting Framework packages.

9 Find out more and more areas of application

9.1 Immediate user installation

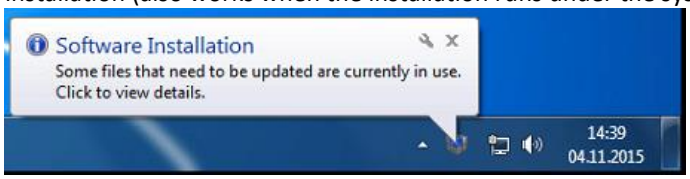
The user configuration (InstallUser.ps1) is started automatically after installation on all logged on users on the device, if the feature has not been disabled on the configuration. If the feature has been disabled, it is possible to manually start the task from the EXE file:
C:\Windows_ScriptingFramework\Modul\ActiveSetup.exe

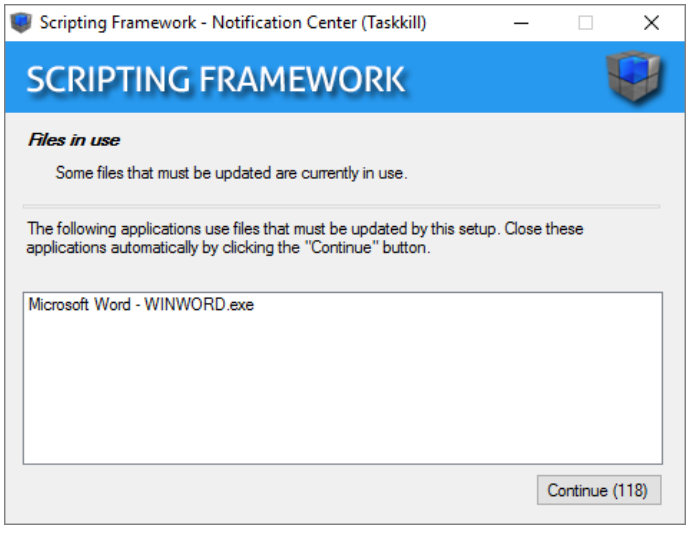


When logging on, the order in which the user settings are applied can be influenced. The Priority value for the user settings can be written in Install.ps1:
SF-Register32 "HKEY_LOCAL_MACHINE" "SOFTWARE\ScriptingFramework\UserSettings\%_PkgIdentifier%" "Priority" "1" "REG_SZ". Priority 1 is the highest.

9.2 Notification Center (user notifications)

The Scripting Framework Notification Center gives you the opportunity to interact with the user during an installation (also works when the installation runs under the system account). Example of an interaction:



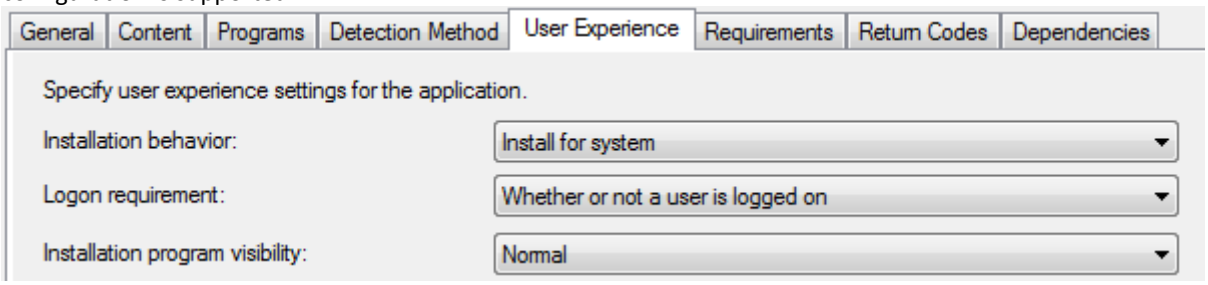


The interface currently supports the languages German, English, Italian and French and is expandable via language.xml. When issuing a notification, the Notification Center is automatically displayed in the respective language of the user's system.

If the language is not stored in XML, the interface is displayed default in English. The title logo can be replace with your own picture (C:\Windows_ScriptingFramework\Modul\NotificationCenterLogo.png).

Default Logo (480x50):

In connection with SCCM does the notification regardless of the "User Experience" settings, also this configuration is supported:



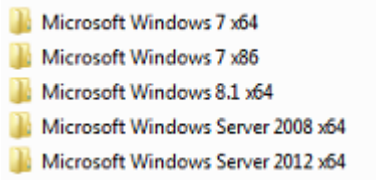
The following commands support currently a notification:

- SF-ShowInstallNotification
- SF-Taskkill

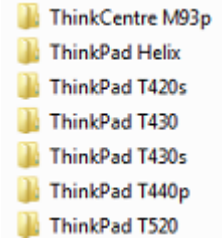
9.3 Installing drivers

We have a finished Scripting Framework package for easy integration of drivers. Upon request we provide it at your disposal.

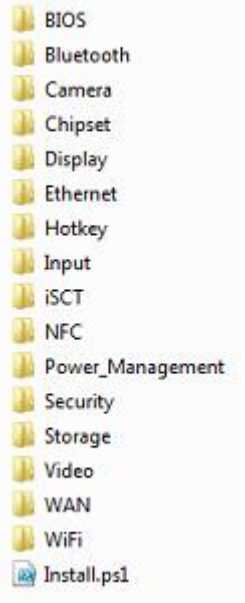
The package itself can PnP driver installing and also perform setup routines. We use it also for BIOS updates and writing BIOS settings. The package resolves automatically the driver path of each hardware model. For a share we used the following structure:



Structure under „Microsoft Windows 8.1 x64“:



Structure under Thinkpad Helix:



In the Install.ps1 all commands of Scripting Framework can be used as usual. Example:

```
# === Intel Bluetooth Driver ==========================================
f_Run "%_SystemDrive%\Drivers\Bluetooth\Setup.exe" "/qn" -Wait -NoErrors

# === Broadcom NFC Driver =============================================
f_Run "%_WindowsSystem64%\pnputil.exe" "-i -a ""%_SystemDrive%\Drivers\NFC\GFWN04WW\*.inf""" -Wait -NoErrors

# === Video Intel Driver ==============================================
f_Run "%_WindowsSystem64%\pnputil.exe" "-i -a ""%_SystemDrive%\Drivers\Video\GFDAS2WW\DisplayAudio\*.inf""" -Wait -NoErrors
f_Run "%_WindowsSystem64%\pnputil.exe" "-i -a ""%_SystemDrive%\Drivers\Video\GFDAS2WW\Graphics\*.inf""" -Wait -NoErrors

# === Hotkey ==========================================================
f_Run "%_SystemDrive%\Drivers\Hotkey\SETUP.EXE" "/S" -Wait -NoErrors

# === WAN Driver ======================================================
f_Run "%_SystemDrive%\Drivers\WAN\setup.exe" "-S" -Wait -NoErrors

# === Cleaning ========================================================
f_RD  "%_SystemDrive%\Intel"
```

9.4 Load licensed fonts only for the user session (Citrix)

With the function *SF-FontInstall* you have with the parameter -CurrentSessionLoad the possibility to load fonts only for the current session. After a logout from the user, the fonts are not longer available. Together with the "Scripting Mode" we use this function successfully.