

opsi-winst / opsi-script Handbuch (4.12.0)

Inhaltsverzeichnis

1	Copyright	1
2	opsi-winst / opsi-script reference card (4.12.0.x)	2
2.1	Global text constants	2
2.1.1	System directories	2
2.1.1.1	System directories [W]:	2
2.1.2	Common (AllUsers) directories [W]:	2
2.1.3	Default User directories [W]:	3
2.1.4	Current user directories [W]:	3
2.1.5	/AllNtUserProfiles directory constants [W]:	3
2.1.6	opsi-winst Path and Directories [W/L]:	3
2.1.7	Network informations [W/L]:	4
2.1.8	Service Data [W/L]	4
2.2	In Primary Sections	4
2.2.1	Kinds of Primary Sections [W/L]:	4
2.2.2	Winst control [W/L]:	5
2.2.3	Variables [W/L]:	5
2.2.3.1	Strings	5
2.2.3.2	Stringlists	6
2.2.4	Functions	6
2.2.4.1	String functions	6
2.2.4.2	String list functions	8
2.2.4.3	Boolean operators and functions	10
2.2.4.4	Misc functions	11
2.2.4.5	Flow control	11
2.3	Secondary Sections	12
2.3.1	Winbatch [W/L]	12
2.3.2	DosBatch and DosInAnIcon (ShellBatch and ShellInAnIcon) [W/L]	13
2.3.3	ExecWith [W/L]	13
2.3.4	Files [W/L]	14

2.3.5	Registry [W]	15
2.3.6	Patches [W/L]	15
2.3.7	PatchTextFile [W/L]	16
2.3.8	LinkFolder [W/L]	17
2.3.9	OpsiServiceCall [W/L]	18
2.3.10	PatchHosts [W/L]	18
2.3.11	XMLPatch [W]	18
2.3.12	ExecPython [W/L]	18
2.3.13	LdapSearch [W]	19
2.4	By Topic	19
2.4.1	Compare related functions [W/L]	19
2.4.2	Crypt / Hash related functions [W/L]	19
2.4.3	Defined Functions and Libraries [W/L]	19
2.4.4	Encoding related functions [W/L]	20
2.4.5	Error / Warning related functions [W/L]	21
2.4.6	File related functions [W/L]	21
2.4.7	Ini file related functions [W/L]	22
2.4.8	Interaction [W/L]	22
2.4.9	License Management related functions [W/L]	22
2.4.10	Logging related functions [W/L]	23
2.4.11	Network related functions [W/L]	24
2.4.12	Number related functions [W/L]	24
2.4.13	Operating System related functions [W/L]	24
2.4.14	opiservicecall and json Related functions [W/L]	24
2.4.15	opsi related functions [W/L]	25
2.4.16	Process and Script Related functions [W/L]	25
2.4.17	Registry related functions [W]	26
2.4.18	String handling functions [W/L]	27
2.4.19	Stringlist handling functions [W/L]	28
2.4.20	Time / Date related functions [W/L]	29
2.4.21	Usercontext / loginscripts related functions [W]:	29
3	Einführung	30
4	Betrieb unter Linux (<i>opsi-script</i>)	31
4.1	Einführung	31
4.2	Wichtige Unterschiede und Hinweise	31
4.3	Pfade	31
4.4	Verwendung von Pfaden in opsi-script	32

4.5	Linuxspezifische Funktionen	32
4.6	Beispiel Scripte für Linux	32
4.6.1	Run on Linux only	32
4.6.2	Which Linux Version	33
4.6.3	ShellInAnIcon call	34
4.6.4	Add a repository	34
4.6.5	Delete a repository	36
4.6.6	Installing a package	37
4.6.7	Integrated Example	38
5	Start und Aufrufparameter	39
5.1	Logging Dateien und Pfade	40
5.2	Zentrale Konfiguration über opsi Configs (Host Parameter)	41
6	Weitere Konfigurationsoptionen	42
6.1	Zentrales Protokollieren von Fehlermeldungen	42
6.2	Skinnable <i>opsi-winst/opsi-script</i> [W/L]	42
6.3	<i>opsi-winst/opsi-script</i> encoding [W/L]	42
7	Das <i>opsi-winst/opsi-script</i> Skript	44
7.1	Ein Beispiel	44
7.2	Primäre und sekundäre Unterprogramme des <i>opsi-winst/opsi-script</i> Skripts	45
7.3	String-Ausdrücke im <i>opsi-winst/opsi-script</i> Skript	46
8	Definition und Verwendung von Variablen und Konstanten im <i>opsi-winst/opsi-script</i> Skript	47
8.1	Allgemeines	47
8.2	Globale Textkonstanten [W/L]	48
8.2.1	Verwendung	48
8.2.2	Beispiel	48
8.2.3	Systemverzeichnis	48
8.2.3.1	Basissystemverzeichnis [W]	48
8.2.3.2	Gemeinsames (AllUsers) Verzeichnis [W]	48
8.2.3.3	Default User Verzeichnis [W]	49
8.2.3.4	Aktuelles (logged in oder usercontext) User Verzeichnis [W]	49
8.2.3.5	/AllNtUserProfiles Verzeichnis Konstanten [W]	49
8.2.4	<i>opsi-winst/opsi-script</i> Pfad und Verzeichnisse [W/L]	50
8.2.5	Netzwerk Informationen [W/L]	51
8.2.6	Service Daten [W/L]	51
8.3	String- (oder Text-) Variable [W/L]	52
8.3.1	Deklaration	52
8.3.2	Wertzuweisung	52
8.3.3	Verwendung von Variablen in String-Ausdrücken	53
8.3.4	Sekundäre und Primäre Sektion im Vergleich	53
8.4	Variable für String-Listen [W/L]	53

9	Syntax und Bedeutung der primären Sektionen eines <i>opsi-winst/opsi-script</i> Skripts [W/L]	55
9.1	Die primären Sektionen [W/L]	55
9.2	Parametrisierungsanweisungen für den <i>opsi-winst/opsi-script</i> [W/L]	56
9.2.1	Beispiel	56
9.2.2	Festlegung der Protokollierungstiefe	56
9.2.3	Benötigte <i>opsi-winst/opsi-script</i> Version [W/L]	57
9.2.4	Reaktion auf Fehler [W/L]	57
9.2.5	Vordergrund [W]	58
9.2.6	Fenster Modus / Skin / Aktivitätsanzeige	59
9.3	String-Werte, String-Ausdrücke und String-Funktionen [W/L]	59
9.3.1	Elementare String-Werte	60
9.3.2	Strings in Strings („geschachtelte“ String-Werte)	60
9.3.3	String-Verknüpfung	60
9.3.4	String-Ausdrücke	60
9.3.5	String-Funktionen zur Ermittlung des Betriebssystemtyps	61
9.3.6	String-Funktionen zur Ermittlung von Umgebungs- und Aufrufparametern [W/L]	62
9.3.7	Werte aus der Windows-Registry lesen und für sie aufbereiten [W]	62
9.3.8	Werte aus Ini-Dateien lesen [W/L]	64
9.3.9	Produkt Properties auslesen [W/L]	64
9.3.10	Informationen aus etc/hosts entnehmen [W/L]	65
9.3.11	String-Verarbeitung [W/L]	65
9.3.12	Weitere String-Funktionen	71
9.3.13	(String-) Funktionen für die Lizenzverwaltung [W/L]	78
9.3.14	Abrufen der Fehlerinformationen von Serviceaufrufen [W/L]	79
9.4	String-Listenverarbeitung [W/L]	79
9.4.1	Info Maps	80
9.4.2	Erzeugung von String-Listen aus vorgegebenen String-Werten [W/L]	88
9.4.3	Laden der Zeilen einer Textdatei in eine String-Liste	89
9.4.4	(Wieder-) Gewinnen von Einzelstrings aus String-Listen oder Dateien [W/L]	89
9.4.5	String-Listen-Erzeugung mit Hilfe von Sektionsaufrufen	90
9.4.6	String-Listen aus der Registry [W]	91
9.4.7	String-Listen aus Produkt Properties [W/L]	93
9.4.8	Sonstige String-Listen [W/L]	94
9.4.9	Transformation von String-Listen [W/L]	94
9.4.10	Iteration durch String-Listen [W/L]	99
9.5	opiservicecall und JSON Funktionen [W/L]	99
9.6	Umgang mit Zahlen [W/L]	103
9.7	Process und Script Funktionen [W/L]	106
9.8	Spezielle Kommandos [W/L]	109

9.9	Kommandos zur Steuerung des Logging [W/L]	110
9.10	Anweisungen für Information und Interaktion [W/L]	112
9.11	Commands for userLoginScripts / User Profile Management [W]	112
9.12	For-To Schleife [W/L]	113
9.13	Switch / Case Statement [W/L]	115
9.14	Bedingungsanweisungen (if-Anweisungen) [W/L]	116
9.14.1	Allgemeine Syntaxregeln [W/L]	117
9.14.2	Boolesche Ausdrücke [W/L]	117
9.15	Include Kommandos [W/L]	122
9.15.1	Include Kommandos: Syntax	123
9.15.2	Include Kommandos: Library	124
9.16	Aufrufe von Unterprogrammen [W/L]	126
9.16.1	Komponenten eines Unterprogrammaufrufs	126
9.17	Reboot-Steueranweisungen	128
9.18	Scriptabbruch und fehlgeschlagene Installation anzeigen [W/L]	129
9.19	Lokale Funktionen [W/L]	131
9.19.1	Konzept	131
9.19.2	Syntax	132
9.19.3	Beispiele	132
9.20	Import von Library Funktionen [W/L]	136
10	Sekundäre Sektionen	137
10.1	Files-Sektionen [W/L]	137
10.1.1	Beispiele	137
10.1.2	Aufrufparameter (Modifier) [W]	137
10.1.3	Kommandos [W/L]	138
10.2	Patches-Sektionen [W/L]	141
10.2.1	Beispiele	142
10.2.2	Aufrufparameter	142
10.2.3	Kommandos	143
10.3	PatchHosts-Sektionen [W/L]	144
10.4	IdapiConfig-Sektionen	145
10.5	PatchTextFile-Sektionen [W/L]	145
10.5.1	Aufrufparameter	145
10.5.2	Kommandos	146
10.5.3	Beispiele	147
10.6	LinkFolder-Sektionen [W/L]	147
10.6.1	LinkFolder-Sektionen in Windows	148
10.6.2	Beispiele	149

10.6.3	LinkFolder-Sektionen in Linux	150
10.7	XMLPatch-Sektionen [W]	151
10.7.1	Aufrufparameter	151
10.7.2	Struktur eines XML-Dokuments	151
10.7.3	Optionen zur Bestimmung eines Sets von Elementen	153
10.7.4	Patch-Aktionen	154
10.7.5	Rückgaben an das aufrufende Programm	155
10.7.6	Beispiele	155
10.8	ProgmanGroups-Sektionen	155
10.9	WinBatch-Sektionen [W/L]	156
10.9.1	Aufrufparameter (Modifier)	156
10.9.2	Beispiele	159
10.10	DOSBatch/DosInAnIcon (ShellBatch/ShellInAnIcon) Sektionen [W/L]	159
10.10.1	Aufrufparameter	160
10.10.2	Einfangen der Ausgaben	162
10.10.3	Beispiele	163
10.11	Registry-Sektionen [W]	163
10.11.1	Beispiele	163
10.11.2	Aufrufparameter	163
10.11.3	Kommandos	164
10.11.4	Registry-Sektionen, die alle NTUser.dat patchen	166
10.11.5	Registry-Sektionen im Regedit-Format	167
10.11.6	Registry-Sektionen im AddReg-Format	168
10.12	OpsiServiceCall Sektion [W/L]	168
10.12.1	Aufrufparameter	169
10.12.2	Sektionsformat	170
10.12.3	Beispiele	171
10.13	ExecPython Sektionen [W/L]	171
10.13.1	Verflechten eines Python Skripts mit einem opsi-winst/opsi-script Skript	172
10.13.2	Beispiele	172
10.14	ExecWith Sektionen [W/L]	172
10.14.1	Aufrufparameter (Modifier)	173
10.14.2	Weitere Beispiele	174
10.15	LDAPsearch Sektion [W]	174
10.15.1	LDAP – Protokoll, Service, Verzeichnis	174
10.15.2	LDAPsearch Aufrufparameter	176
10.15.3	Einengung der Suche	177
10.15.4	LDAPsearch Sektion Syntax	178
10.15.5	Beispiele	179

11 64 Bit-Unterstützung unter Windows [W]	181
12 Kochbuch	185
12.1 Löschen einer Datei in allen Userverzeichnissen	185
12.2 Überprüfen, ob ein spezieller Service läuft	186
12.3 Skript für Installationen im Kontext eines lokalen Administrators	187
12.4 XML-Datei patchen: Setzen des Vorlagenpfades für OpenOffice.org 2.0	194
12.5 XML-Konfiguration für eine MsSql-Anwendung patchen: Ein Beispiel mit irreführend benannten Attributen	195
12.6 XML-Datei einlesen mit dem opsi-winst	196
12.7 Einfügen einer Namensraumdefinition in eine XML-Datei	197
12.8 Herausfinden, ob ein Skript im Kontext eines bestimmten Events läuft	198
13 Spezielle Fehlermeldungen	199
14 opsi documentation generator	200
14.1 Introduction	200
14.2 opsi-doc-generator program	200
14.3 opsi-doc-generator marker	201
14.4 opsi-doc-generator examples	202
15 opsi-winst Übungen	203
15.1 Einführung	203
15.2 Erstellung von Winst-Scripten	203
15.2.1 1. Übung	203
15.2.2 2. Übung	203
15.2.3 3. Übung	204
15.2.4 4. Übung	204
15.2.5 5. Übung	204
15.3 Lösungen	204
15.3.1 Lösung Übung 1	204
15.3.2 Lösung Übung 2	205
15.3.3 Lösung Übung 3	205
15.3.4 Lösung Übung 4	206
15.3.5 Lösung Übung 5	206
16 opsi-script libraries	208
16.1 opsi-script libraries von uib	208
16.1.1 Documentation of opsi library: uib_backend.opsiscript	208
16.1.1.1 Documentation of local function setProductsToSetup	208
16.1.1.2 Documentation of local function getInstalledLocalbootProducts	209

16.1.1.3	Documentation of local function <code>delOpsiPoc</code>	210
16.1.1.4	Documentation of local function <code>backupOpsiPoc</code>	210
16.1.1.5	Documentation of local function <code>restoreOpsiPoc</code>	211
16.1.1.6	Documentation of local function <code>getInstalledLocalbootProductsWithVersion</code>	212
16.1.1.7	Documentation of local function <code>getInstallableLocalbootProductsWithVersion</code>	212
16.1.2	Documentation of opsi library: <code>uib_bootutils.opsiscript</code>	213
16.1.2.1	Documentation of local function <code>delFromWindowsBootmanager</code>	213
16.1.2.2	Documentation of local function <code>getWinBcdbootGuid</code>	214
16.1.2.3	Documentation of local function <code>getUefiBcdbootGuid</code>	214
16.1.2.4	Documentation of local function <code>bootNextToWinLabel</code>	215
16.1.2.5	Documentation of local function <code>bootNextToUefiLabel</code>	215
16.1.2.6	Documentation of local function <code>getDiskUuid</code>	216
16.1.2.7	Documentation of local function <code>enablePEPartition</code>	217
16.1.3	Documentation of opsi library: <code>uib_exitcode.opsiscript</code>	218
16.1.3.1	Documentation of local function <code>isMsiExitcodeFatal</code>	218
16.1.3.2	Documentation of local function <code>isMsExitcodeFatal_short</code>	219
16.1.3.3	Documentation of local function <code>isAdvancedMsiExitcodeFatal</code>	220
16.1.3.4	Documentation of local function <code>isInnoExitcodeFatal</code>	221
16.1.3.5	Documentation of local function <code>isInstallshieldExitcodeFatal</code>	221
16.1.3.6	Documentation of local function <code>isNsisExitcodeFatal</code>	222
16.1.4	Documentation of opsi library: <code>uib_strlistutils.opsiscript</code>	223
16.1.4.1	Documentation of local function <code>compareLists</code>	223

Kapitel 1

Copyright

Das Copyright an diesem Handbuch liegt bei der uib gmbh in Mainz.

Dieses Handbuch ist veröffentlicht unter der creative commons Lizenz *Namensnennung - Weitergabe unter gleichen Bedingungen* (by-sa).



Eine Beschreibung der Lizenz finden Sie hier:

<http://creativecommons.org/licenses/by-sa/3.0/de/>

Der rechtsverbindliche Text der Lizenz ist hier:

<http://creativecommons.org/licenses/by-sa/3.0/de/legalcode>

Die Software von opsi ist in weiten Teilen Open Source.

Nicht Open Source sind die Teile des Quellcodes, welche neue Erweiterungen enthalten die noch unter Kofinanzierung stehen, also noch nicht bezahlt sind.

siehe auch: <http://uib.de/de/opsi-erweiterungen/erweiterungen/>

Der restliche Quellcode ist veröffentlicht unter der GPLv3:



Der rechtsverbindliche Text der GPLv3 Lizenz ist hier:

<http://www.gnu.org/licenses/agpl-3.0-standalone.html>

Deutsche Infos zur GPLv3: <http://www.gnu.org/licenses/agpl-3.0.de.html>

Für Lizenzen zur Nutzung von opsi im Zusammenhang mit Closed Source Software kontaktieren Sie bitte die uib gmbh.

Die Namen *opsi*, *opsi.org*, *open pc server integration* und das opsi-logo sind eingetragene Marken der uib gmbh.

Kapitel 2

opsi-winst / opsi-script reference card (4.12.0.x)

2.1 Global text constants

2.1.1 System directories

2.1.1.1 System directories [W]:

`%ProgramFilesDir%`: *c:\program files*

`%ProgramFiles32Dir%`: *c:\Program Files (x86)* //since 4.10.8

`%ProgramFiles64Dir%`: *c:\program files* //since 4.10.8

`%ProgramFilesSysnativeDir%`: *c:\program files* //since 4.10.8

`%Systemroot%`: *c:\windows*

`%System%`: *c:\windows\system32*

`%Systemdrive%`: *c:*

`%ProfileDir%`:

NT5: *c:\Documents and Settings*

NT6: *C:\users*

see also: [Abschnitt 8.2.3.1](#)

2.1.2 Common (AllUsers) directories [W]:

`%AllUsersProfileDir%` or `%CommonProfileDir%`:

NT5: *c:\Documents and Settings\All Users*

NT6: *C:\Users\Public*

`%CommonStartMenuPath%` or `%CommonStartmenuDir%`:

NT5: *c:\Documents and Settings\All Users\Startmenu*

NT6: *C:\ProgramData\Microsoft\Windows\Start Menu*

`%CommonAppdataDir%`:

NT5: *c:\Documents and Settings\All Users\Application Data*

NT6: *C:\ProgramData*

`%CommonDesktopDir%`

NT5: *c:\Documents and Settings\All Users\Desktop*

NT6: *C:\Users\Public\Desktop*

`%CommonStartupDir%`

NT5: *c:\Documents and Settings\All Users\Autostart*

NT6: *C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp*

`%CommonProgramsDir%`

see also : Abschnitt [8.2.3.2](#)

2.1.3 Default User directories [W]:

`%DefaultUserProfileDir%` //since 4.11.1.1

see also : Abschnitt [8.2.3.3](#)

2.1.4 Current user directories [W]:

User is the logged in user or given by /usercontext.

`%AppdataDir%` or `%CurrentAppdataDir%` : //since 4.10.8.13

NT5: *c:\Documents and Settings\%USERNAME%\Application Data* NT6: *c:\users\%USERNAME%\Appdata\Roaming*

`%CurrentStartmenuDir%` //since 4.10.8.13

`%CurrentDesktopDir%` //since 4.10.8.13

`%CurrentStartupDir%` //since 4.10.8.13

`%CurrentProgramsDir%` //since 4.10.8.13

`%CurrentSendToDir%` //since 4.10.8.13

`%CurrentProfileDir%` //since 4.11.2.1

see also : Abschnitt [8.2.3.4](#)

2.1.5 /AllNtUserProfiles directory constants [W]:

`%UserProfileDir%`

or

`%CurrentProfileDir%` // since 4.11.2.1

NT5: *c:\Documents and Settings\%USERNAME%*

NT6: *c:\users\%USERNAME%*

see also : Abschnitt [8.2.3.5](#)

2.1.6 opsi-winst Path and Directories [W/L]:

`%ScriptPath%` or `%ScriptDir%`

`%ScriptDrive%`

`%WinstDir%`

`%WinstVersion%` //4.10.8.3

`%Logfile%`

`%opsiScriptHelperPath%` `%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib` // since 4.11.3.2

`%opsiTmpDir%` : c:\opsi.org\tmp // since 4.11.4.3
`%opsiLogDir%` : c:\opsi.org\log // since 4.11.4.3
`%opsidata%` : c:\opsi.org\data // since 4.12.0.12
`%opsiapplog%` : c:\opsi.org\applog // since 4.12.0.12
see also : Abschnitt [8.2.4](#)

2.1.7 Network informations [W/L]:

`%Host%` : value of environment variable HOST.
`%PCName%`: value of environment variable PCNAME, or if absent of COMPUTERNAME.
`%Username%` : Name of actual user.
`%IPName%` : The dns name of the pc. Usually identical with the netbios name and therefore with `%PCName%` besides that the netbios names uses to be uppercase.
`%IPAddress%` : may be the IP-Address of the machine. Use funktion `GetMyIpByTarget()` instead.
see also : [\[GetMyIpByTarget\]](#)
see also : Abschnitt [8.2.5](#)

2.1.8 Service Data [W/L]

`%HostID%` : FQDN of the client
`%opsiserviceURL%`
`%opsiServer%`
`%opsiDepotId%` //since 4.11.4
`%opsiserviceUser%` FQDN used for the connection to the opsi-config-server
`%opsiservicePassword%`
`%installingProdName%`: productid //since 4.10.8
`%installingProdVersion%`: product version //since 4.10.8
`%installingProduct%` : productid (deprecated)
see also : Abschnitt [8.2.6](#)

2.2 In Primary Sections

2.2.1 Kinds of Primary Sections [W/L]:

[Initial]
[Actions]
[sub<identifier>]
sub <file name>
[ProfileActions] [W]
see also : Kapitel [9](#)

2.2.2 Winst control [W/L]:

`encoding=<encoding> // (default is system encoding) since 4.11.4.2 see also : \[encoding\]`

`LogLevel (deprecated) see also : Abschnitt 9.2.2`

`SetLogLevel = <number> or SetLogLevel = <string> // (default=6) see also : \[SetLogLevel\]`

```
SetLogLevel = 7
SetLogLevel = "7"
```

`ExitOnError = <boolean value> // (default=false) see also : \[ExitOnError\]`

`ScriptErrorMessages = <boolean value> // (default=true) see also : \[ScriptErrorMessages\]`

see also : [\[opsi-script-configs_ScriptErrorMessages\]](#)

`FatalOnSyntaxError = <boolean value> // (default=true) since 4.11.3.2 see also : \[FatalOnSyntaxError\]`

`FatalOnRuntimeError = <boolean value> // (default=false) since 4.11.3.2 see also : \[FatalOnRuntimeError\]`

`AutoActivityDisplay = <boolean value> // (default=false); if true shows a marquee (endless) progressbar while winbatch/dosbatch sections are . //since 4.11.4.7 see also : \[AutoActivityDisplay\] see also : \[opsi-script-configs_AutoActivityDisplay\]`

`Message <string> or Message = <const string> see also : \[Message\]`

`ShowMessageFile <string> see also : \[ShowMessageFile\]`

`ShowBitMap [<file name>] [<sub title>] see also : \[ShowBitMap\]`

`comment <string> or comment = <const string> see also : \[comment\]`

`LogError <string> or LogError = <const string> see also : \[LogError\]`

`LogWarning <string> or LogWarning = <const string> see also : \[LogWarning\]`

`includelog <file name> <tail size> //since 4.11.2.1 [W/L] see also : \[includelog\]`

`includelog <file name> <tail size> [<encoding>] //since 4.11.4.1 [W/L] see also : \[includelog\]`

```
includelog "%Scriptpath%\test-files\10lines.txt" "5"
```

`SetConfidential <secret string> //since 4.11.3.5 [W/L] see also : \[SetConfidential\]`

`asConfidential(<secret string expression>) //since 4.12.0.16 [W/L] see also : \[asConfidential\]`

`Pause <string> or Pause = <const string> see also : \[Pause\]`

`Stop <string> or stop = <const string> see also : \[Stop\]`

`include_insert <file name> // since 4.11.3 see also : \[include_insert\]`

`include_append <file name> // since 4.11.3 see also : \[include_append\]`

`NormalizeWinst // (set normal window state) since 4.11.3 see also : \[NormalizeWinst\]`

`IconizeWinst // (set minimized window state) see also : \[IconizeWinst\]`

`MaximizeWinst // (set maximized window state) // since 4.11.5.1 see also : \[MaximizeWinst\]`

`RestoreWinst // (restore last window state) see also : \[RestoreWinst\]`

`SetSkinDirectory <path to skin.ini> // since 4.11.3.5 see also : \[SetSkinDirectory\]`

2.2.3 Variables [W/L]:

2.2.3.1 Strings

`DefVar <variable name>`

`Set <variable name> = <value>`

see also : Abschnitt [8.3](#)

2.2.3.2 Stringlists

DefstringList <variable name>

see also : Abschnitt 8.4

2.2.4 Functions

2.2.4.1 String functions

GetOS // *Linux* or *Windows_NT* [W/L] see also : [\[GetOS\]](#)

getLinuxDistroType // *debian* or *redhat* or *suse* (see [getLinuxVersionMap](#)) [L] see also : [\[getLinuxDistroType\]](#)

GetMsVersionInfo //Windows Version Information [W] see also : [\[GetMsVersionInfo\]](#)

GetSystemType //OS Architecture ("64 Bit System" or "x86 System") [W/L] see also : [\[GetSystemType\]](#)

getRegistryValue(<keystr>, <varstr> [, <access str>]) : string //since 4.12.0.16 [W]
<access str> = one of 32bit, 64bit, sysnative ; default sysnative see also : [\[getRegistryValue\]](#)

GetRegistrystringvalue ("[key] var") [W] see also : [\[GetRegistrystringvalue\]](#)

GetRegistryStringValue32 ("[key] var") //since 4.10.8 [W] see also : [\[GetRegistryStringValue32\]](#)

GetRegistryStringValue64 ("[key] var") //since 4.10.8 [W] see also : [\[GetRegistryStringValue64\]](#)

GetRegistryStringValueSysNative ("[key] var") //since 4.10.8 [W] see also : [\[GetRegistryStringValueSysNative\]](#)

GetValueFromInifile (file, section, key, default value) [W/L]

```
GetValueFromInifile("myfile", "mysec", "mykey", "")
```

see also : [\[GetValueFromInifile\]](#)

GetProductProperty (<PropertyName>, <DefaultValue>) [W/L] see also : [\[GetProductProperty\]](#)

GetConfidentialProductProperty (<PropertyName>, <DefaultValue>) //since 4.11.5.2 [W/L] see also : [\[GetConfidentialProductProperty\]](#)

trim(<string>) [W/L] see also : [\[trim\]](#)

lower(<string>) [W/L] see also : [\[lower\]](#)

upper(<string>) [W/L] see also : [\[upper\]](#)

unquote(<string>, <quote-string>) //since 4.11.2.1 [W/L] see also : [\[unquote\]](#)

unquote2(<string>, <quote-string>) //since 4.11.5.2 [W/L] see also : [\[unquote2\]](#)

stringReplace(<string>, <oldPattern>, <newPattern>) //since 4.11.3 [W/L] see also : [\[stringReplace\]](#)

strLength(<string>) //since 4.11.3 [W/L] see also : [\[strLength\]](#)

strPos(<string>, <sub string>) //since 4.11.3 [W/L] see also : [\[strPos\]](#)

strPart(<string>, <start pos>, <number of chars>) //since 4.11.3 [W/L] see also : [\[strPart\]](#)

getValue(<key string>, <hash string list>) [W/L] see also : [\[getValue\]](#)

getValueBySeparator(<key string>, <separator string>, <hash string list>) //since 4.11.2.1 [W/L] see also : [\[getValueBySeparator\]](#)

getValueFromFile(<key string>, <file name>) //since 4.11.4.4 [W/L] see also : [\[getValueFromFile\]](#)

getValueFromFileBySeparator(<key string>, <separator string>, <file name>) //since 4.11.4.4 [W/L] see also : [\[getValueFromFileBySeparator\]](#)

getLastExitCode : string (exitcode) [W/L] see also : [\[getLastExitCode\]](#)

DemandLicenseKey(poolId [, productId [, windowsSoftwareId]])

```
set $mykey$ = DemandLicenseKey ("", "office2007")
```

see also : [\[DemandLicenseKey\]](#)

```
FreeLicense (`poolId [, productId [,windowsSoftwareId]])`
```

```
set $result$ = FreeLicense("", "office2007")
```

see also : [\[FreeLicense\]](#)

GetUserSID(<Windows Username>) see also : [\[GetUserSID\]](#)

GetLoggedInUser //since 4.11.1.2 see also : [\[GetLoggedInUser\]](#)

GetUsercontext //since 4.11.1.2 see also : [\[getLastExitCode\]](#)

GetScriptMode possible values *Machine,Login* //since 4.11.2.1 see also : [\[GetUsercontext\]](#)

saveVersionToProfile - save productversion-packageversion to local profile //since 4.11.2.1 see also : [\[saveVersionToProfile\]](#)

readVersionFromProfile : string - read productversion-packageversion from local profile //since 4.11.2.1 see also : [\[readVersionFromProfile\]](#)

scriptWasExecutedBefore : boolean - is true if saved and running productversion-packageversion are identical //since 4.11.2.1 see also : [\[scriptWasExecutedBefore\]](#)

GetHostsName (<hostaddress>) [W/L] see also : [\[GetHostsName\]](#)

GetHostsAddr (<hostname>) [W/L] see also : [\[GetHostsAddr\]](#)

ExtractFilePath (<path>) [W/L] see also : [\[ExtractFilePath\]](#)

calculate(<arithmetic string expression>) // since 4.11.3.5 : knows: +-*/() [W/L] see also : [\[calculate\]](#)

DecStrToHexStr (<decstring>, <hexlength>) [W/L] see also : [\[DecStrToHexStr\]](#)

HexStrToDecStr (<hexstring>) [W/L] see also : [\[HexStrToDecStr\]](#)

base64EncodeStr(<string>) [W/L] see also : [\[base64EncodeStr\]](#)

base64DecodeStr(<string>) [W/L] see also : [\[base64DecodeStr\]](#)

convert2Jsonstr(<string>) //since 4.10.8.3

RandomStr [W/L] see also : [\[RandomStr\]](#)

CompareDotSeparatedStrings(<string1>, <string2>) : string [W/L] see also : [\[CompareDotSeparatedStrings_str\]](#)

CompareDotSeparatedNumbers(<string1>, <string2>) : string [W/L] see also : [\[CompareDotSeparatedNumbers_str\]](#)

EnvVar (<environment variable>) [W/L] see also : [\[EnvVar\]](#)

ParamStr [W/L] see also : [\[ParamStr\]](#)

getDiffTimeSec (Time in seconds since last marktime) //since 4.11.3 [W/L] see also : [\[getDiffTimeSec\]](#)

SidToName(<well known sid>) //since 4.11.3: gives localized name of the sid [W] see also : [\[SidToName\]](#)

GetMyIpByTarget(<target ip addr>) : string //since 4.11.3.2 /4.11.6 [W/L] see also : [\[GetMyIpByTarget\]](#)

GetIpByName(<ip addr / ip name>) //since 4.11.3.2 [W/L] see also : [\[GetIpByName\]](#)

reencodestr(<str>, <from>, <to>) //since 4.11.4.2 [W/L] see also : [\[reencodestr\]](#)

strLoadTextFile (<filename>) //since 4.11.4.6 [W/L] see also : [\[strLoadTextFile\]](#)

strLoadTextFileWithEncoding (<filename>, <encoding>) //since 4.11.4.6 [W/L] see also : [\[strLoadTextFileWithEncoding\]](#)

GetShortWinPathName(<longpath string>) //since 4.11.5.2 [W] see also : [\[GetShortWinPathName\]](#)

GetNtVersion Deprecated - please use GetMsVersionInfo [W] see also : [\[GetMsVersionInfo\]](#)

IniVar (<key>) : (deprecated; use GetProductProperty) [W] see also : [\[GetProductProperty\]](#)

SubstringBefore (<string1>, <string2>) (deprecated; use splitString / takestring) [W/L] see also : [\[splitString\]](#)

2.2.4.2 String list functions

splitString (<string1>, <string2>) [W/L]

```
set $list1$ = splitString ("\\server\share\dir", "\")
```

see also : [\[splitString\]](#)

splitStringOnWhiteSpace (<string>) [W/L] see also : [\[splitStringOnWhiteSpace\]](#)

loadTextFile (<file name>) [W/L] see also : [\[loadTextFile\]](#)

loadUnicodeTextFile (<file name>) [W] see also : [\[loadUnicodeTextFile\]](#)

loadTextFileWithEncoding (<file name> , <encoding>) //since 4.11.5 [W/L] see also : [\[loadTextFileWithEncoding\]](#)

composeString (<string list>, <Link>) [W/L] see also : [\[composeString\]](#)

takeString (<index>, <list>) [W/L] see also : [\[takeString\]](#)

setStringInListAtIndex (<newstring>, <list>, <indexstr>) : stringlist //since 4.11.6 [W/L] see also : [\[setStringInListAtIndex\]](#)

takeFirstStringContaining (<list>, <search string>) [W/L] see also : [\[takeFirstStringContaining\]](#)

getOutputStreamFromSection (<dos section name>) [W/L]

```
set $list$= getOutputStreamFromSection ('DosInAnIcon_try')
```

see also : [\[getOutputStreamFromSection\]](#)

shellCall (<command string>) : stringlist (output) //since 4.11.4.2 [W/L]

```
set $list$= shellCall('net start')
```

see also : [\[shellCall_list\]](#)

getReturnListFromSection (<xml section name>) [W/L] see also : [\[getReturnListFromSection\]](#)

getListContaining (<list>, <search string>) [W/L] see also : [\[getListContaining\]](#)

getListContainingList (<list1>, <list2>) //since 4.11.3.7 [W/L] see also : [\[getListContainingList\]](#)

count (<list>) [W/L] see also : [\[count\]](#)

emptylist (<list>) //since 4.11.3.7 [W/L] see also : [\[emptylist\]](#)

for %<identifier>% in <list> do <one statement | sub section> [W/L]

```
for %s% in $list1$ do sub_test_string
```

see also : [\[forInDo\]](#)

GetProcessList //since 4.11.1.2; gives list of exename;pid;dom/user [W/L] see also : [\[GetProcessList\]](#)

getProductPropertyList (<proprname>, <default value>) //since 4.11.3 [W/L] see also : [\[getProductPropertyList\]](#)

getRegistryKeyList32 (<regkey>) //since 4.11.3 [W] see also : [\[getRegistryKeyList32\]](#)

getRegistryKeyList64 (<regkey>) //since 4.11.3 [W] see also : [\[getRegistryKeyList64\]](#)

getRegistryKeyListSysnative (<regkey>) //since 4.11.3 [W] see also : [\[getRegistryKeyListSysnative\]](#)

getRegistryVarList32 (<regkey>) //since 4.11.3 [W] see also : [\[getRegistryVarList32\]](#)

getRegistryVarList64 (<regkey>) //since 4.11.3 [W] see also : [\[getRegistryVarList64\]](#)

getRegistryVarListSysnative (<regkey>) //since 4.11.3 [W] see also : [\[getRegistryVarListSysnative\]](#)

getProfilesDirList //since 4.11.3.2 [W/L] see also : [\[getProfilesDirList\]](#)

GetLocaleInfoMap [W] see also : [\[GetLocaleInfoMap\]](#)

GetMSVersionMap [W] see also : [\[GetMSVersionMap\]](#)

getLinuxVersionMap //since 4.11.4 [L]

keys are (example):

```
Distributor ID=Ubuntu
Description=Ubuntu 12.04.2 LTS
Release=12.04
Codename=precise
kernel name=Linux
node name=detlefv05
kernel release=3.2.0-40-generic-pae
kernel version=#64-Ubuntu SMP Mon Mar 25 21:44:41 UTC 2013
machine=i686
processor=athlon
hardware platform=i386
operating system=GNU/Linux
```

see also : [\[getLinuxVersionMap\]](#)

`getFileInfoMap(<file name>)` [W] see also : [\[getFileInfoMap\]](#)

`getProductMap` // since 4.11.2.4 [W/L]

keys are: id, name, description, advice, productversion, packageversion, priority, installationstate, lastactionrequest, lastactionresult, installedversion, installedpackage, installedmodificationtime,actionrequest

see also : [\[getProductMap\]](#)

`getRegistryVarMap32(<regkey>)` //since 4.11.3 [W] see also : [\[getRegistryVarMap32\]](#)

`getRegistryVarMap64(<regkey>)` //since 4.11.3 [W] see also : [\[getRegistryVarMap64\]](#)

`getRegistryVarMapSysnative(<regkey>)` //since 4.11.3 [W] see also : [\[getRegistryVarMapSysnative\]](#)

`getHWBiosInfoMap` //since 4.11.4 [W/L] see also : [\[getHWBiosInfoMap\]](#)

`createStringList (<string0>, <string1> ,...)` [W/L]

```
set $list1$ = createStringList ('a','b')
```

see also : [\[createStringList\]](#)

`reverse (<list>)` [W/L] see also : [\[reverse\]](#)

`getSectionNames(<ini-file>)` [W/L] see also : [\[getSectionNames\]](#)

`retrieveSection (<section name>)` [W/L] see also : [\[retrieveSection\]](#)

`getSubList (<start index> : <end index>, <list>)` [W/L] see also : [\[getSubList\]](#)

`getSubListByMatch (<search string>, <target list>) :stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByMatch_sl\]](#)

`getSubListByMatch (<search list>, <target list>) :stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByMatch_ll\]](#)

`getSubListByContaining (<search string>, <target list>) :stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByContaining_sl\]](#)

`getSubListByContaining (<search list>, <target list>) :stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByContaining_ll\]](#)

`getSubListByKey (<search string>, <target list>) :stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByKey_sl\]](#)

`getSubListByKey (<search list>, <target list>) :stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByKey_ll\]](#)

`getKeyList (<list>) :stringlist` //since 4.12.0.14 [W/L] see also : [\[getKeyList\]](#)

`addtolist(<list>,<string>)` //since 4.10.8 [W/L] see also : [\[addtolist\]](#)

`addListToList(<dest list>,<src list>)` //since 4.10.8 [W/L] see also : [\[addListToList\]](#)

`reencodestrlist(<list>, <from>, <to>)` //since 4.11.4.2 [W/L] see also : [\[reencodestrlist\]](#)

`removeFromListByContaining(<search string>, <target list>)` : stringlist //since 4.11.5.1 [W/L] see also : [\[removeFromListByContaining_str\]](#)

`removeFromListByContaining(<search list>, <target list>)` : `stringlist` //since 4.11.5.1 [W/L] see also : [\[removeFromListByContaining_list\]](#)

`removeFromListByMatch(<searchstring>,<target list>)` : `stringlist` //since 4.11.6 [W/L] see also : [\[removeFromListByMatch\]](#)

2.2.4.3 Boolean operators and functions

see also : Abschnitt [9.14.2](#)

`<string1> = <string2>` [W/L]

`<bool1> AND <bool2>` [W/L]

`<bool1> OR <bool2>` [W/L]

`NOT(<bool3>)` [W/L]

`FileExists (<file name>)` [W/L] see also : [\[FileExists\]](#)

`FileExists32 (<file name>)` [W] see also : [\[FileExists\]](#)

`FileExists64 (<file name>)` [W] see also : [\[FileExists\]](#)

`FileExistsSysNative (<file name>)` [W] see also : [\[FileExists\]](#)

`LineExistsIn (<string>, <file name>)` [W/L] see also : [\[LineExistsIn\]](#)

`LineBeginning_ExistsIn (<string>, <file name>)` [W/L] see also : [\[LineBeginning_ExistsIn\]](#)

`LineContaining_ExistsIn (<string>, <file name>)` //since 4.11.4.10: true: if a in <file name> contains <string> [W/L] see also : [\[LineContaining_ExistsIn\]](#)

`XMLAddNamespace(<XMLfilename>, <XMLelementname>, <XMLnamespace>)` [W] see also : [\[XMLAddNamespace\]](#)

`XMLRemoveNamespace(<XMLfilename>, <XMLelementname>, <XMLnamespace>)` [W] see also : [\[XMLRemoveNamespace\]](#)

`HasMinimumSpace (<drive letter>, <capacity>)` [W] see also : [\[HasMinimumSpace\]](#)

Example:

```
if not (HasMinimumSpace ("%SYSTEMDRIVE%", "500 MB"))
    LogError "Required free space of 500 MB not available on %SYSTEMDRIVE%"
    isFatalError
endif
```

`opsiLicenseManagementEnabled` [W/L] see also : [\[opsiLicenseManagementEnabled\]](#)

`runningAsAdmin` //since 4.11.1.1 [W/L] see also : [\[runningAsAdmin\]](#)

`isLoginScript` //since 4.11.2.1 [W] see also : [\[isLoginScript\]](#)

`contains(<str>, <substr>)` : `bool` //since 4.11.3: true if <substr> in <str> [W/L] see also : [\[contains\]](#)

`isNumber(<str>)` //since 4.11.3: true if <str> represents an integer [W/L] see also : [\[isNumber\]](#)

`runningOnUefi` //since 4.11.4.3: true: if the running OS was booted in UEFI mode [W] see also : [\[runningOnUefi\]](#)

`runningInPE` //since 4.12.0.13: true: if the running OS is a Windows PE [W/L] see also : [\[runningInPE\]](#)

`isDriveReady(<drive letter>)` //since 4.11.4.4: true: if the drive can be accessed [W] see also : [\[isDriveReady\]](#)

`saveTextFile(<list>, < filename>)` //since 4.11.4.4: true: if list is successfully written to file [W/L] see also : [\[saveTextFile\]](#)

`saveTextFileWithEncoding(<list>, < filename>, <encoding>)` : `bool` //since 4.11.6.4: true: if list is successfully written to file [W/L] see also : [\[saveTextFileWithEncoding\]](#)

`CompareDotSeparatedNumbers(<str1>,<relation str>,<str2>)` //since 4.11.5.2: [W/L] see also : [\[CompareDotSeparatedNumbers_bool\]](#)

`CompareDotSeparatedStrings(<str1>,<relation str>,<str2>)` //since 4.11.5.2: [W/L] see also : [\[CompareDotSeparatedStrings_bool\]](#)

`RegKeyExists(<regkey>[,<access str>])` : `bool` //since 4.12.0.16 [W]
<access str> = one of 32bit, 64bit, sysnative ; default sysnative see also : [\[RegKeyExists\]](#)

`RegVarExists(<regkey>, <var str> [,<access str>])` : `bool` //since 4.12.0.16 [W]
<access str> = one of 32bit, 64bit, sysnative ; default sysnative see also : [\[RegVarExists\]](#)

2.2.4.4 Misc functions

`Killtask <process name>` [W/L] see also : [\[Killtask\]](#)

`requiredWinstVersion <relation operator> <version>` [W/L] see also : [\[requiredWinstVersion\]](#)

```
requiredWinstVersion >= "4.10"
```

`UpdateEnvironment` //since 4.11.5 [W]:

Subsequent calls of `winbatch` with the parameter `/RunElevated` will see the changed Environment (NT6 only). see also : [\[UpdateEnvironment\]](#)

2.2.4.5 Flow control

`if - else - endif` [W/L] see also : [\[IfElseEndif\]](#)

Syntax:

```
if <condition>
;statement(s)
[else
;statement(s)]
endif
```

Example:

```
Set $NTVer$ = GetMsVersionInfo
if ( $NTVer$ >= "6" )
    sub_install_win7
else
    if ( $NTVer$ = "5.1" )
        sub_install_winXP
    else
        stop "not a supported OS-Version"
    endif
endif
```

`for - to - do Statement` //since 4.11.5 [W/L] see also : [\[ForToDo\]](#)

`for %<temporary string variable>% = <start string> to <end string> do <one statement>`

Example:

```
for %s% = "1" to "5" do sub_iteration_test
```

`Switch / Case Statement` //since 4.11.5 [W/L] see also : [\[SwitchCase\]](#)

Syntax:

```

Switch <string expression>
  Case <string const>
    <statement (s)>
  EndCase
  [ DefaultCase
    <statement (s)>
    EndCase ]
EndSwitch

```

Example:

```

set $ConstTest$ = "5"
Switch $ConstTest$
  Case "1"
    set $CompValue$ = "1"
  EndCase
  Case "2"
    set $CompValue$ = "2"
  EndCase
  DefaultCase
    set $CompValue$ = "notexisting"
  EndCase
EndSwitch

```

isFatalError [W/L] see also : [\[isFatalError\]](#)

isFatalError <string> //since 4.11.3.2 [W/L] see also : [\[isFatalError\]](#)

isSuccess //since 4.11.3.7 [W/L] see also : [\[isSuccess\]](#)

isSuspended //since 4.11.4.1 [W/L] see also : [\[isSuspended\]](#)

noUpdateScript //since 4.11.3.7 [W/L] see also : [\[noUpdateScript\]](#)

ExitWindows /Reboot [W/L] see also : [\[Reboot\]](#)

ExitWindows /ImmediateReboot [W/L] see also : [\[ImmediateReboot\]](#)

ExitWindows /ImmediateLogout [W] see also : [\[ImmediateLogout\]](#)

ExitWindows /ShutdownWanted [W] see also : [\[ShutdownWanted\]](#)

ExitWindows /RebootWanted (deprecated, acts like /Reboot) [W] see also : [\[Reboot\]](#)

sleepSeconds <Integer> or <string> : noresult [W/L] see also : [\[sleepSeconds\]](#)

ChangeDirectory <directory> //since 4.11.2.6 [W/L] see also : [\[ChangeDirectory\]](#)

2.3 Secondary Sections

2.3.1 Winbatch [W/L]

see also : Abschnitt [10.9](#)

Function: execute programs via operating system API

[WinBatch<identifier>]

Modifier:

/LetThemGo

```

/WaitForProcessEnding "<program.exe>"
/TimeoutSeconds <seconds>
/WaitForWindowAppearing <window title> (does not work with 64 Bit programs) [W]
/WaitForWindowVanish <window title> (does not work with 64 Bit programs) [W]
/RunElevated // since 4.11.3: only at >= NT6 ; no network access [W]
/RunAsLoggedOnUser // since 4.11.3.5 ; works only inside userLoginScripts [W]
/32Bit //since 4.11.3.5 [W]
/64Bit //since 4.11.3.5 [W]
/SysNative //since 4.11.3.5 [W]

```

2.3.2 DosBatch and DosInAnIcon (ShellBatch and ShellInAnIcon) [W/L]

see also : Abschnitt [10.10](#)

Function: Execute section via cmd.exe

```

[DosBatch<identifier>] <optional parameters> <winst <modifier>>
[DosInAnIcon<identifier>] <optional parameters> <winst <modifier>>
[ShellBatch<identifier>] <optional parameters> <winst <modifier>>
[ShellInAnIcon<identifier>] <optional parameters> <winst <modifier>>

```

Modifier: //since 4.11.1.1

```

/32Bit [W]
/64Bit [W]
/SysNative [W]
/showoutput [W/L] // since 4.11.4.7

```

The modifiers has to be seperated by *winst* from the parameters.

```
DosInAnIcon_do_64bit_stuff winst /64Bit
```

Commands: see manual

2.3.3 ExecWith [W/L]

see also : Abschnitt [10.14](#)

Function: Execute section via any interpreter

```
[ExecWith<identifier>] <path to interpreter>
```

Modifier:

```

/LetThemGo
/EscapeStrings
/32Bit //since 4.11.3.5 [W]
/64Bit //since 4.11.3.5 [W]
/SysNative //since 4.11.3.5 [W]

```

The modifiers has to be seperated by *winst* from the parameters. The following example call the 64Bit version of the powershell.exe.

```
ExecWith_do_64bit_stuff "%System%\WindowsPowerShell\v1.0\powershell.exe" winst /64Bit
```

Commands: see manual

2.3.4 Files [W/L]

see also : Abschnitt [10.1](#)

Function: File Operations

[Files<identifier>]

Modifier [W]:

/AllNTUserProfiles

/AllNTUserSendTo [W]

/32Bit //since 4.10.8 [W]

/64Bit //since 4.10.8 [W]

/SysNative //since 4.10.8 [W]

Commands:

checkTargetPath = <destination directory> [W/L]

copy [Options] <source file(s)> <destination directory> [W/L]

some options:

-s recursive [W/L]

-V version control against targetdir [W]

-v version control against targetdir, %systemroot% and %system% (**do not use it**) [W]

-c continue without reboot even if it is needed [W]

-d date check [W]

-u update [W]

-x extract [W]

-w weak (do not overwrite protected files) [W]

-n no overwrite [W]

-r copy read only attribute [W]

-h follow symlinks [L] //since 4.11.6.14

delete [Options] <path[/mask]] // [W/L]

some options: -s rekursiv -f force

Example (**do not forget the trailing Backslash**):

```
delete -sf c:\delete_this_dir\
```

del [Options] <path[/mask]] //since 4.11.2.1 [W/L]

Works like delete but on

```
del -s -f c:\not-exists
```

if c:\not-exists not exists it do not search complete c:\ for not-exists

Example (**you may forget the trailing Backslash**):

```
del -sf c:\delete_this_dir
```

chmod <mask> <path> //since 4.11.4.1 [L]

hardlink <existing file> <new file> // since 4.11.5 [W/L]

symlink <existing file> <new file> // since 4.11.5 [W/L]

At Windows symlink is only available at NT6 and up.

rename <old filename> <new filename> // since 4.11.5 [W/L]

move <old filename> <new filename> // since 4.11.5 [W/L]

2.3.5 Registry [W]

see also : Abschnitt [10.11](#)

Function: edit Registry

Standard method call:

```
[Registry<identifier>]
```

works with the specified section.

Alternative method call:

```
Registry loadUnicodeTextFile(<.reg file>) /regedit
```

import the specified <.reg file>.

Alternative method call (deprecated):

```
Registry loadUnicodeTextFile(<.addreg file>) /addreg
```

import the specified <.addreg file>.

Modifier:

```
/AllNTUserDats
```

```
/32Bit //since 4.10.8
```

```
/64Bit //since 4.10.8
```

```
/SysNative //since 4.10.8
```

Commands:

```
OpenKey <Key>
```

```
openkey [HKLM\Software\opsi.org]
```

```
Set <varname> = <registry type>:<value>
```

```
Add <varname> = <registry type>:<value>
```

Examples for registry types:

```
set "var1" = "my string"
set "var2" = REG_SZ:"my string"
set "var3" = REG_EXPAND_SZ:"%ProgramFiles%"
set "var4" = REG_DWORD:123          (Decimal)
set "var5" = REG_DWORD:0x7b       (Hexadecimal)
set "var6" = REG_BINARY:00 01 02 0F 10
set "var7" = REG_MULTI_SZ:"A|BC|de"
```

```
Supp <varname> <list char> <supplement>
```

```
supp "Path" ; "C:\utils; %JAVABIN%"
```

```
GetMultiSZFromFile <varname> <file name>
```

```
SaveValueToFile <varname> <file name>
```

```
DeleteVar <varname>
```

```
DeleteKey <registry key> (does since 4.11.2.1 also work with /AllNTUserDats)
```

2.3.6 Patches [W/L]

see also : Abschnitt [10.2](#)

Function: edit Ini-files

[Patches<identifier>] <file name>

Modifier:

/AllNTUserProfiles //since 4.11.3 [W]

Commands:

add [<section name>] <variable1> = <value1>

set [<section name>] <variable1> = <value1>

addnew [<section name>] <variable1> = <value1>

change [<section name>] <variable1> = <value1>

del [<section name>] <variable1> = <value1>

del [<section name>] <variable1>

delsec [<section name>]

replace <variable1>=<value1> <variable2>=<value2>

2.3.7 PatchTextFile [W/L]

see also : Abschnitt [10.5](#)

Function: edit text files

[PatchTextFile<identifier>] <file name>

Modifier:

/AllNTUserProfiles //since 4.11.3.4 [W]

Commands:

Set_Mozilla_Pref ("<preference type>", "<preference key>", "<preference value>")

preference types are usally:

pref, user_pref, lock_pref

AddStringListElement_To_Mozilla_Pref ("<preference type>", "<preference key>", "<add value>")

Set_Netscape_User_Pref ("<key>", "<value>") (*deprecated*)

AddstringListElement_To_Netscape_User_Pref (*deprecated*)

FindLine <search string>

FindLine_StartingWith <search string>

FindLine_Containing <search string>

GoToTop

AdvanceLine [<number of lines>]

GoToBottom

DeleteTheLine

AddLine_ <line> or Add_Line_ <line>

InsertLine <line> or Insert_Line_ <line>

AppendLine <line> or Append_Line <line>

Append_File <file name>

Subtract_File <file name>

SaveToFile <file name>

Sorted

setKeyValueSeparator <separator char> //since 4.11.4.4 [W/L]

setValueByKey <keystr> <valuestr> //since 4.11.4.4 [W/L]

2.3.8 LinkFolder [W/L]

see also : Abschnitt [10.6](#)

Function: Startmenue + Desktop Icons

[LinkFolder<identifier>]

Commands:

`set_basefolder` <system folder>

`set_subfolder` <folder path> (at Linux set always "")

```
set_link
  name:          <link name>
  target:        <path and name of the program>
  parameters:    [command line arguments]
  working_dir:   [working directory]
  icon_file:     [path and name of icon file , default=target]
  icon_index:    [number of icon in icon file , default=0] [W]
  shortcut:     [keyboard shortcut for calling the target] [W]
  link_categories: [list of categories] [L]
end_link
```

`delete_element` <link name>

`delete_subfolder` <folder path> [W]

The predefined virtual system folders which can be used are at Windows:

`desktop`, `sendto`, `startmenu`, `startup`, `programs`, `desktopdirectory`,
`common_startmenu`, `common_programs`, `common_startup`, `common_desktopdirectory`
 and at Linux:

`common_programs`, `common_startup`, `desktop`, `startup`

Predefined `link_categories` for Linux:

AudioVideo, Audio, Video, Development, Education, Game, Graphics, Network, Office, Settings, System, Utility

Examples

```
set_basefolder common_desktopdirectory
set_subfolder ""
set_link
  name: opsi-winst
  target: "%ProgramFiles32Dir%\opsi.org\opsi-client-agent\opsi-winst\winst32.exe"
end_link
```

```
[LinkFolder_configed_lin]
set_basefolder common_programs
set_subfolder ""

set_link
  name: opsi-configed-Local
  target: java
  parameters: $parameter$
  icon_file: "$InstallDir$/opsi.png"
  link_categories: System;Utility;
end_link
```

The predefined virtual system folders:

`desktop`, `sendto`, `startmenu`, `startup`, `programs`, `desktopdirectory`

are pointing to the folders of the user that the script is running. If you use it in a userLoginScript with the opsi *User Profile Management* extension these virtual folders point to the folder of the user that just had logged in.

shortcut defaults to empty. // since 4.11.6.7

shortcut may be a combination of [*shift,alt,ctrl*] (not case sensitiv) divided by ' , ' -, + an a *Key* or a *Virtual Key Code*. The *Key* is a letter (*A - Z*) or a numeral (*0 - 9*). All other Keys must be given by there *Virtual Key Code* identifier. To get these identifier (as well as the allowed combinations) just use the following helper program:

<http://download.uib.de/opsi4.0/helper/showkeys.exe>

2.3.9 OpsiServiceCall [W/L]

see also : Abschnitt [10.12](#)

Function: opsi-Service access

[OpsiServiceCall<identifier>]

Commands: see manual

2.3.10 PatchHosts [W/L]

see also : Abschnitt [10.3](#)

Function: hosts-files bearbeiten

[PatchHosts<identifier>]

Commands:

setaddr <hostname> <IPaddress>

setname <IPaddress> <hostname>

setalias <hostname> <alias>

setalias <IPadresse> <alias>

delalias <hostname> <alias>

delalias <IPaddress> <alias>

delhost <hostname>

delhost <ipadresse>

setComment <ident> <comment>

2.3.11 XMLPatch [W]

see also : Abschnitt [10.7](#)

Function: edit XML files

[XMLPatch<identifier>]

Commands: see manual

2.3.12 ExecPython [W/L]

see also : Abschnitt [10.13](#)

Function: Execute section via python interpreter

[ExecPython<identifier>]

Commands: see manual

2.3.13 LdapSearch [W]

see also : Abschnitt [10.15](#)

Function: read from LDAP

[LdapSearch<identifier>]

Commands: see manual

2.4 By Topic

2.4.1 Compare related functions [W/L]

CompareDotSeparatedStrings(<string1>, <string2>) : string [W/L] see also : [\[CompareDotSeparatedStrings_str\]](#)

CompareDotSeparatedStrings(<str1>,<relation str>,<str2>) : bool //since 4.11.5.2: [W/L] see also : [\[CompareDotSeparatedStrings_bool\]](#)

CompareDotSeparatedNumbers(<string1>, <string2>) : string [W/L] see also : [\[CompareDotSeparatedNumbers_str\]](#)

CompareDotSeparatedNumbers(<str1>,<relation str>,<str2>) : bool //since 4.11.5.2: [W/L] see also : [\[CompareDotSeparatedNumbers_bool\]](#)

boolToString(<boolean expression>) : bool string (true/false) // since 4.12.0.0 [W/L] see also : [\[boolToString\]](#)

stringToBool(<string expression: true/false>) : boolean // since 4.12.0.0 [W/L] see also : [\[stringToBool\]](#)

2.4.2 Crypt / Hash related functions [W/L]

DecStrToHexStr (<decstring>, <hexlength>) : string [W/L] see also : [\[DecStrToHexStr\]](#)

HexStrToDecStr (<hexstring>) : string [W/L] see also : [\[HexStrToDecStr\]](#)

base64EncodeStr(<string>) : string [W/L] see also : [\[base64EncodeStr\]](#)

base64DecodeStr(<string>) : string [W/L] see also : [\[base64DecodeStr\]](#)

RandomStr : string [W/L] see also : [\[RandomStr\]](#)

encryptStringBlow(<keystring>,<datastring>) : string [W/L] see also : [\[encryptStringBlow\]](#)

decryptStringBlow(<keystring>,<datastring>) : string [W/L] see also : [\[decryptStringBlow\]](#)

md5sumFromFile(<path to file>) : string [W/L] see also : [\[md5sumFromFile\]](#)

2.4.3 Defined Functions and Libraries [W/L]

since 4.12.0.0

Definition

```
DefFunc <func name>([calltype parameter ptype][, [calltype parameter ptype]]) : ftype
<function body>
endfunc
```

Where:

- DefFunc is the keyword used to start defining a local function..

- `<func name>` is the freely chosen name of the function.
- `calltype` is the call type of the parameter [`val` | `ref`]. `val`=*Call by Value*, `ref`=*Call by Reference*. Default: `val`
- `parameter` is the free selected name of the call parameter which is available as a local variable within the function under the aforementioned name.
- `ptype` is the type of data of the parameter whether `string` or `stringlist`.
- `fctype` is the type of data of the function whether `string`, `stringlist` or `void`. `void` declares that no result is returned.
- `<function body>`: is the body of the function which opsi-script syntax must suffice.
- `endfunc` is the keyword used to end defining a local function..

see also : Abschnitt [9.19](#)

```
importLib <string expr> ; import library // since 4.12.0.0
```

```
<string expr> : <file name>[.<file extension>][: :<function name>]
```

If no `.<file extension>` is given `.opsiscript` is used as default.

If no `:<function name>` is given, all function from the given file will be imported.

`<file name>` is:

- A complete path to an existing file. [W/L]
- A existing file in `%ScriptPath%` [W/L]
- A file in `%opsiScriptHelperPath%\lib` [W]
Is equivalent to: `%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib`
- A existing file in `%ScriptPath%/../lib` [W/L]
- A existing file in `%WinstDir%\lib` [W] or `/usr/share/opsi-script/lib` [L]

The tests for the location of the `<file name>` are done in the order above. *opsi-script* uses the first file it finds that has a matching name.

see also : Abschnitt [9.20](#)

2.4.4 Encoding related functions [W/L]

```
encoding=<encoding> // (default is system encoding) since 4.11.4.2 see also : \[encoding\]
```

```
GetLocaleInfoMap : stringlist [W] see also : \[GetLocaleInfoMap\]
```

```
reencodestr(<str>, <from>, <to>) : string //since 4.11.4.2 [W/L] see also : \[reencodestr\]
```

```
reencodestrlist(<list>, <from>, <to>) : stringlist //since 4.11.4.2 [W/L] see also : \[reencodestrlist\]
```

```
loadUnicodeTextFile (<file name>) : stringlist [W] see also : \[loadUnicodeTextFile\]
```

```
loadTextFileWithEncoding (<file name> , <encoding>) : stringlist //since 4.11.5 [W/L] see also : \[loadTextFileWithEncoding\]
```

```
strLoadTextFileWithEncoding (<filename> , <encoding>) : string //since 4.11.4.6 [W/L] see also : \[strLoadTextFileWithEncoding\]
```

```
saveTextFileWithEncoding(<list>, < filename>, <encoding>) : bool //since 4.11.6.4: true: if list is successfully written to file [W/L] see also : \[saveTextFileWithEncoding\]
```

```
includeLog <file name> <tail size> [<encoding>] ` : noresult` //since 4.11.4.1 [W/L] see also : \[includeLog\]
```

see also : Abschnitt [6.3](#)

2.4.5 Error / Warning related functions [W/L]

`ExitOnError` = <boolean value> // (default=false) see also : [\[ExitOnError\]](#)

`ScriptErrorMessages` = <boolean value> // (default=true)

see also : [\[ScriptErrorMessages\]](#)

see also : [\[opsi-script-configs_ScriptErrorMessages\]](#)

`FatalOnSyntaxError` = <boolean value> // (default=true) since 4.11.3.2 see also : [\[FatalOnSyntaxError\]](#)

`FatalOnRuntimeError` = <boolean value> // (default=false) since 4.11.3.2 see also : [\[FatalOnRuntimeError\]](#)

`LogError` <string> or `LogError` = <const string> see also : [\[LogError\]](#)

`LogWarning` <string> or `LogWarning` = <const string> see also : [\[LogWarning\]](#)

`isFatalError` [W/L] see also : [\[isFatalError\]](#)

`isFatalError` <string> //since 4.11.3.2 [W/L] see also : [\[isFatalError\]](#)

`markErrorNumber` see also : [\[markErrorNumber\]](#)

`errorsOccurredSinceMark` <relation> <integer> : boolean see also : [\[errorsOccurredSinceMark\]](#)

```
markErrorNumber
comment "log error and thereby increase the error counter"
if errorsOccurredSinceMark > 0
    comment "There was an error ..."
endif
```

`getLastExitCode` : string (exitcode) [W/L] see also : [\[getLastExitCode\]](#)

`shellCall` (<command string>) : string (exitcode) //since 4.11.6.1 [W/L] see also : [\[shellCall_str\]](#)

`processCall`(<string>) : string (exitcode) //since 4.11.6.1 [W/L] see also : [\[processCall\]](#)

`getLastServiceErrorClass` : string see also : [\[getLastServiceErrorClass\]](#)

`getLastServiceErrorMessage` : string see also : [\[getLastServiceErrorMessage\]](#)

2.4.6 File related functions [W/L]

`strLoadTextFile` (<file name>) : string [W/L] see also : [\[strLoadTextFile\]](#)

`strLoadTextFileWithEncoding` (<filename> , <encoding>) : string //since 4.11.4.6 [W/L] see also : [\[strLoadTextFileWithEncoding\]](#)

`loadTextFile` (<file name>) : stringlist [W/L] see also : [\[loadTextFile\]](#)

`loadUnicodeTextFile` (<file name>) : stringlist [W] see also : [\[loadUnicodeTextFile\]](#)

`loadTextFileWithEncoding`(<file name> , <encoding>) : stringlist //since 4.11.5 [W/L] see also : [\[loadTextFileWithEncoding\]](#)

`FileExists` (<file name>) : bool [W/L] see also : [\[FileExists\]](#)

`FileExists32` (<file name>) : bool [W] see also : [\[FileExists\]](#)

`FileExists64` (<file name>) : bool [W] see also : [\[FileExists\]](#)

`FileExistsSysNative` (<file name>) : bool [W] see also : [\[FileExists\]](#)

`LineExistsIn` (<string>, <file name>) : bool [W/L] see also : [\[LineExistsIn\]](#)

`LineBeginning_ExistsIn` (<string>, <file name>) : bool [W/L] see also : [\[LineBeginning_ExistsIn\]](#)

`LineContaining_ExistsIn`(<string>, <file name>) : bool //since 4.11.4.10: true: if a in <file name> contains <string> [W/L] see also : [\[LineContaining_ExistsIn\]](#)

`saveTextFile(<list>, <filename>)` : `bool` //since 4.11.4.4: true: if list is successfully written to file [W/L] see also : [\[saveTextFile\]](#)

`saveTextFileWithEncoding(<list>, <filename>, <encoding>)` : `bool` //since 4.11.6.4: true: if list is successfully written to file [W/L] see also : [\[saveTextFileWithEncoding\]](#)

`getFileInfoMap(<file name>)` : `stringlist` [W] see also : [\[getFileInfoMap\]](#)

`getFileInfoMap32(<file name>)` : `stringlist` //since 4.11.6.6 [W] see also : [\[getFileInfoMap\]](#)

`getFileInfoMap64(<file name>)` : `stringlist` //since 4.11.6.6 [W] see also : [\[getFileInfoMap\]](#)

`getFileInfoMapSysnative(<file name>)` : `stringlist` //since 4.11.6.6 [W] see also : [\[getFileInfoMap\]](#)

`ExtractFilePath (<path>)` : `string` [W/L] see also : [\[ExtractFilePath\]](#)

see also: Abschnitt [2.3.4](#)

see also: Abschnitt [2.3.7](#)

2.4.7 Ini file related functions [W/L]

`GetValueFromInifile (file, section, key, default value)` : `string` [W/L]

```
GetValueFromInifile("myfile", "mysec", "mykey", "")
```

see also : [\[GetValueFromInifile\]](#)

`getSectionNames(<ini-file>)` : `stringlist` [W/L] see also : [\[getSectionNames\]](#)

`retrieveSection (<section name>)` : `stringlist` [W/L] see also : [\[retrieveSection\]](#)

`getValue(<key string>, <hash string list>)` : `string` [W/L] see also : [\[getValue\]](#)

`getValueBySeparator(<key string>, <separator string>, <hash string list>)` : `string` //since 4.11.2.1 [W/L] see also : [\[getValueBySeparator\]](#)

`getValueFromFile(<key string>, <file name>)` : `string` //since 4.11.4.4 [W/L] see also : [\[getValueFromFile\]](#)

`getValueFromFileBySeparator(<key string>, <separator string>, <file name>)` : `string` //since 4.11.4.4 [W/L] see also : [\[getValueFromFileBySeparator\]](#)

see also: Abschnitt [2.3.6](#)

2.4.8 Interaction [W/L]

`Pause <string>` or `Pause = <const string>` see also : [\[Pause\]](#)

`Stop <string>` or `stop = <const string>` see also : [\[Stop\]](#)

`setActionProgress <string>` : `noresult` //since 4.11.3 [W/L] see also : [\[setActionProgress\]](#)

`Message <string>` or `Message = <const string>` see also : [\[Message\]](#)

`ShowMessageFile <string>` see also : [\[ShowMessageFile\]](#)

`ShowBitMap [<file name>] [<sub title>]` see also : [\[ShowBitMap\]](#)

2.4.9 License Management related functions [W/L]

`DemandLicenseKey(poolId [, productId [,windowsSoftwareId]])` : `string`

```
set $mykey$ = DemandLicenseKey ("", "office2007")
```

see also : [\[DemandLicenseKey\]](#)

FreeLicense (`poolId [, productId [,windowsSoftwareId]]) : string`

```
set $result$ = FreeLicense("", "office2007")
```

see also : [\[FreeLicense\]](#)

getLastServiceErrorClass : string see also : [\[getLastServiceErrorClass\]](#)

getLastServiceErrorMessage : string see also : [\[getLastServiceErrorMessage\]](#)

- opsiLicenseManagementEnabled : bool see also : [\[opsiLicenseManagementEnabled\]](#)

2.4.10 Logging related functions [W/L]

SetLogLevel = <number> or SetLogLevel = <string> // (default=6)

```
SetLogLevel = 7
SetLogLevel = "7"
```

see also : [\[SetLogLevel\]](#) see also : [\[opsi-script-configs_default_loglevel\]](#) see also : [\[opsi-script-configs_force_min_loglevel\]](#)

Message <string> or Message = <const string> see also : [\[Message\]](#)

comment <string> or comment = <const string> see also : [\[comment\]](#)

LogError <string> or LogError = <const string> see also : [\[scriptWasExecutedBefore\]](#)

LogWarning <string> or LogWarning = <const string> see also : [\[LogError\]](#)

includelog <file name> <tail size> //since 4.11.2.1 [W/L] see also : [\[includelog\]](#)

includelog <file name> <tail size> [<encoding>] //since 4.11.4.1 [W/L] see also : [\[includelog\]](#)

```
includelog "%Scriptpath%\test-files\10lines.txt" "5"
```

SetConfidential <secret string> //since 4.11.3.5 [W/L] see also : [\[SetConfidential\]](#)

asConfidential(<secret string expression>) //since 4.12.0.16 [W/L] see also : [\[asConfidential\]](#)

opsi-configs

see also : Abschnitt [5.2](#)

opsi-script.global.debug_prog : boolean ; if false: only Warnings and Errors from program logging; default: false
see also : [\[opsi-script-configs_debug_prog\]](#)

opsi-script.global.debug_lib : boolean ; if false: only Warnings and Errors from library logging; default: false
see also : [\[opsi-script-configs_debug_lib\]](#)

opsi-script.global.default_loglevel : intstr ; set the default log level; default: 6
see also : [\[opsi-script-configs_default_loglevel\]](#)

opsi-script.global.force_min_loglevel : intstr ; set the minimal loglevel; default: 0
see also : [\[opsi-script-configs_force_min_loglevel\]](#)

opsi-script.global.ScriptErrorMessages : boolean ; overwrites the opsi-script internal default; default: false
see also : [\[opsi-script-configs_ScriptErrorMessages\]](#)

opsi-script.global.AutoActivityDisplay : boolean ; overwrites the opsi-script internal default; default: true
see also : [\[opsi-script-configs_AutoActivityDisplay\]](#)

2.4.11 Network related functions [W/L]

GetHostsName (<hostaddress>) : string [W/L] see also : [\[GetHostsName\]](#)

GetHostsAddr (<hostname>) : string [W/L] see also : [\[GetHostsAddr\]](#)

GetMyIpByTarget(<target ip addr>) : string //since 4.11.3.2 /4.11.6 [W/L] see also : [\[GetMyIpByTarget\]](#)

GetIpByName(<ip addr / ip name>) : string //since 4.11.3.2 [W/L] see also : [\[GetIpByName\]](#)

2.4.12 Number related functions [W/L]

isNumber(<str>) : bool //since 4.11.3: true if <str> represents an integer [W/L] see also : [\[isNumber\]](#)

CompareDotSeparatedNumbers(<str1>,<relation str>,<str2>) : bool //since 4.11.5.2: [W/L] see also : [\[CompareDotSeparatedNumbers_bool\]](#)

CompareDotSeparatedNumbers(<string1>, <string2>) : string [W/L] see also : [\[CompareDotSeparatedNumbers_str\]](#)

calculate(<arithmetic string expression>) : string (number) // since 4.11.3.5 : knows: +-*/() [W/L] see also : [\[calculate\]](#)

DecStrToHexStr (<decstring>, <hexlength>) : string [W/L] see also : [\[DecStrToHexStr\]](#)

HexStrToDecStr (<hexstring>) : string [W/L] see also : [\[HexStrToDecStr\]](#)

2.4.13 Operating System related functions [W/L]

GetOS : string // *Linux* or *Windows_NT* [W/L] see also : [\[GetOS\]](#)

getLinuxDistroType : string // *debian* or *redhat* or *suse* (see [getLinuxVersionMap](#)) [L] see also : [\[getLinuxDistroType\]](#)

GetMsVersionInfo : string //Windows Version Information [W] see also : [\[GetMsVersionInfo\]](#)

GetMSVersionMap : stringlist [W] see also : [\[GetMSVersionMap\]](#)

getLinuxVersionMap : stringlist //since 4.11.4 [L] see also : [\[getLinuxVersionMap\]](#)

GetSystemType : string //OS Architecture ("64 Bit System" or "x86 System") [W/L] see also : [\[GetSystemType\]](#)

EnvVar (<environment variable>) : string [W/L] see also : [\[EnvVar\]](#)

getProfilesDirList : stringlist //since 4.11.3.2 [W/L] see also : [\[getProfilesDirList\]](#)

runningAsAdmin //since 4.11.1.1 [W/L] see also : [\[runningAsAdmin\]](#)

runningOnUefi //since 4.11.4.3: true: if the running OS was booted in UEFI mode [W] see also : [\[runningOnUefi\]](#)

runningInPE //since 4.12.0.13: true: if the running OS is a Windows PE [W/L] see also : [\[runningInPE\]](#)

isDriveReady(<drive letter>) //since 4.11.4.4: true: if the drive can be accessed [W] see also : [\[isDriveReady\]](#)

2.4.14 opsiservicecall and json Related functions [W/L]

jsonIsValid(<jsonstr>) : boolean //since 4.11.6: [W/L] see also : [\[jsonIsValid\]](#)

jsonIsArray(<jsonstr>) : boolean //since 4.11.6: [W/L] see also : [\[jsonIsArray\]](#)

jsonIsObject(<jsonstr>) : boolean //since 4.11.6: [W/L] see also : [\[jsonIsObject\]](#)

jsonAsObjectHasKey(<jsonstr>,<keystr>) : boolean //since 4.11.6: [W/L] see also : [\[jsonAsObjectHasKey\]](#)

jsonAsArrayCountElements(<jsonstr>) : intstr //since 4.11.6: [W/L] see also : [\[jsonAsArrayCountElements\]](#)

jsonAsObjectCountElements(<jsonstr>) : intstr //since 4.11.6: [W/L] see also : [\[jsonAsObjectCountElements\]](#)

`jsonAsArrayGetElementByIndex(<jsonstr>, <indexstr>)` : `jsonstring` //since 4.11.6: [W/L] see also : [\[jsonAsArrayGetElementByIndex\]](#)

`jsonAsObjectGetValueByKey(<jsonstr>, <keystr>)` : `valuestring` //since 4.11.6: [W/L] see also : [\[jsonAsObjectGetValueByKey\]](#)

`jsonAsObjectSetValueByKey(<jsonstr>, <keystr>,<valuestring>)` : `jsonstring` //since 4.11.6: [W/L] see also : [\[jsonAsObjectSetValueByKey\]](#)

`jsonAsObjectSetStringtypeValueByKey(<jsonstr>, <keystr>,<valuestring>)` : `jsonstring` //since 4.11.6: [W/L] see also : [\[jsonAsObjectSetStringtypeValueByKey\]](#)

`jsonAsObjectDeleteByKey(<jsonstr>, <keystr>)` : `jsonstring` //since 4.11.6.4: [W/L] see also : [\[jsonAsObjectDeleteByKey\]](#)

`jsonAsArrayPutObjectByIndex(<jsonstr>, <indexstr>, <objectstr>)` : `jsonstring` //since 4.11.6: [W/L] see also : [\[jsonAsArrayPutObjectByIndex\]](#)

`jsonAsArrayDeleteObjectByIndex(<jsonstr>, <indexstr>)` : `jsonstring` //since 4.11.6.4: [W/L] see also : [\[jsonAsArrayDeleteObjectByIndex\]](#)

`jsonAsArrayToStringList(<jsonstr>)` : `stringlist` //since 4.11.6: [W/L] see also : [\[jsonAsArrayToStringList\]](#)

`jsonAsObjectGetKeyList(<jsonstr>)` : `stringlist` //since 4.11.6: [W/L] see also : [\[jsonIsArray\]](#)

`jsonStringListToJsonArray(<strlist>)` : `jsonstr` //since 4.11.6: [W/L] see also : [\[jsonAsObjectGetKeyList\]](#)

`convert2Jsonstr(<string>)` //since 4.10.8.3

see also: OpsiServiceCall Abschnitt [2.3.9](#)

see also : Abschnitt [9.5](#)

see also : Abschnitt [10.12](#)

2.4.15 opsi related functions [W/L]

`getProductMap` : `stringlist` // since 4.11.2.4 [W/L]

keys are: id, name, description, advice, productversion, packageversion, priority, installationstate, lastactionrequest, lastactionresult, installedversion, installedpackage, installedmodificationtime,actionrequest see also : [\[getProductMap\]](#)

`getProductPropertyList(<propname>,<default value>)` : `stringlist` //since 4.11.3 [W/L] see also : [\[getProductPropertyList\]](#)

`GetProductProperty (<PropertyName>, <DefaultValue>)` : `string` [W/L] see also : [\[GetProductProperty\]](#)

`GetConfidentialProductProperty (<PropertyName>, <DefaultValue>)` : `string` //since 4.11.5.2 [W/L] see also : [\[GetConfidentialProductProperty\]](#)

`setActionProgress <string>` : `noresult` //since 4.11.3 [W/L] see also : [\[setActionProgress\]](#)

2.4.16 Process and Script Related functions [W/L]

`Killtask <process name> `` : `noresult`` [W/L] see also : [\[Killtask\]](#)

`ChangeDirectory <directory> `` : `noresult`` //since 4.11.2.6 [W/L] see also : [\[ChangeDirectory\]](#)

`GetProcessList` : `stringlist` //since 4.11.1.2; gives list of exename;pid;dom/user [W/L] see also : [\[GetProcessList\]](#)

`processIsRunning(<process name>)` : `boolean` //since 4.11.6.1 [W/L] see also : [\[processIsRunning\]](#)

`shellCall (<command string>)` : `stringlist (output)` //since 4.11.4.2 [W/L] see also : [\[shellCall_list\]](#)

```
set $list$= shellCall('net start')
```

shellCall (<command string>) : noresult //since 4.11.6.1 [W/L] see also : [\[shellCall\]](#)

shellCall (<command string>) : string (exitcode) //since 4.11.6.1 [W/L] see also : [\[shellCall_str\]](#)

powershellcall (<commandstr> [,<access str>=*sysnative* [,<policy bool str>=*true*]]) : stringlist (output) //since 4.12.0.16 [W] see also : [\[powershellCall_list\]](#)

powershellcall (<commandstr> [,<access str>=*sysnative* [,<policy bool str>=*true*]]) : noresult //since 4.12.0.16 [W] see also : [\[powershellCall\]](#)

powershellcall (<commandstr> [,<access str>=*sysnative* [,<policy bool str>=*true*]]) : string (exitcode) //since 4.12.0.16 [W] see also : [\[powershellCall_str\]](#)

getOutputStreamFromSection (<dos section name>) : stringlist (output) [W/L]

```
set $list$= getOutputStreamFromSection ('DosInAnIcon_try')
```

see also : [\[getOutputStreamFromSection\]](#) see also : Abschnitt [10.10.2](#)

processCall(<string>) : string (exitcode) //since 4.11.6.1 [W/L] see also : [\[processCall\]](#)

getLastExitCode : string (exitcode) [W/L] see also : [\[getLastExitCode\]](#)

includelog <file name> <tail size> : noresult //since 4.11.2.1 [W/L] see also : [\[includelog\]](#)

includelog <file name> <tail size> [<encoding>] : noresult //since 4.11.4.1 [W/L] see also : [\[includelog\]](#)

waitForPackageLock(<seconds timeout string>,<bool should we kill>) : bool //since 4.11.6.1 [L] see also : [\[wait-ForPackageLock\]](#)

see also: ExecWith sections Abschnitt [2.3.3](#)

see also: ShellBatch sections Abschnitt [2.3.2](#)

see also: Winbatch sections Abschnitt [2.3.1](#)

2.4.17 Registry related functions [W]

getRegistryValue(<keystr>, <varstr> [, <access str>]) : string //since 4.12.0.16 [W]
<access str> = one of 32bit, 64bit, sysnative ; default sysnative see also : [\[getRegistryValue\]](#)

GetRegistrystringvalue("[key] var") : string [W] see also : [\[GetRegistrystringvalue\]](#)

GetRegistryStringValue32 ("[key] var") : string //since 4.10.8 [W] see also : [\[GetRegistryStringValue32\]](#)

GetRegistryStringValue64 ("[key] var") : string //since 4.10.8 [W] see also : [\[GetRegistryStringValue64\]](#)

GetRegistryStringValueSysNative ("[key] var") : string //since 4.10.8 [W] see also : [\[GetRegistryStringValueSysNative\]](#)

getRegistryKeyList32(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryKeyList32\]](#)

getRegistryKeyList64(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryKeyList64\]](#)

getRegistryKeyListSysnative(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryKeyListSysnative\]](#)

getRegistryVarList32(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryVarList32\]](#)

getRegistryVarList64(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryVarList64\]](#)

getRegistryVarListSysnative(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryVarListSysnative\]](#)

getRegistryVarMap32(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryVarMap32\]](#)

getRegistryVarMap64(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryVarMap64\]](#)

getRegistryVarMapSysnative(<regkey>) : stringlist //since 4.11.3 [W] see also : [\[getRegistryVarMapSysnative\]](#)

`RegKeyExists(<regkey>[,<access str>])` : `bool` //since 4.12.0.16 [W]
 <access str> = one of 32bit, 64bit, sysnative ; default sysnative see also : [\[RegKeyExists\]](#)

`RegVarExists(<regkey>, <var str> [,<access str>])` : `bool` //since 4.12.0.16 [W]
 <access str> = one of 32bit, 64bit, sysnative ; default sysnative see also : [\[RegVarExists\]](#)

see also: Abschnitt [2.3.5](#)

see also : Abschnitt [10.11](#)

see also : Kapitel [11](#)

2.4.18 String handling functions [W/L]

see also : Abschnitt [9.3](#)

`splitString (<string1>, <string2>)` : `stringlist` [W/L] see also : [\[splitString\]](#)

```
set $list1$ = splitString ("\\server\share\dir", "\")
```

`splitStringOnWhiteSpace (<string>)` : `stringlist` [W/L] see also : [\[splitStringOnWhiteSpace\]](#)

`composeString (<string list>, <Link>)` : `string` [W/L] see also : [\[composeString\]](#)

`takeString (<index>, <list>)` : `string` [W/L] see also : [\[takeString\]](#)

`setStringInListAtIndex(<newstring>,<list>,<indexstr>)` : `stringlist` //since 4.11.6 [W/L] see also : [\[setStringInListAtIndex\]](#)

`takeFirstStringContaining(<list>,<search string>)` : `string` [W/L] see also : [\[takeFirstStringContaining\]](#)

`getIndexFromListByContaining(<list> : stringlist,<search string> : string)` : <number> : string` //since 4.12.0.13 [W/L] see also : [\[getIndexFromListByContaining\]](#)

`contains(<str>, <substr>)` : `bool` //since 4.11.3: true if <substr> in <str> [W/L] see also : [\[contains\]](#)

`isNumber(<str>)` : `bool` //since 4.11.3: true if <str> represents an integer [W/L] see also : [\[isNumber\]](#)

`trim(<string>)` : `string` [W/L] see also : [\[trim\]](#)

`lower(<string>)` : `string` [W/L] see also : [\[lower\]](#)

`upper(<string>)` [W/L] see also : [\[upper\]](#)

`unquote(<string>,<quote-string>)` : `string` //since 4.11.2.1 [W/L] see also : [\[unquote\]](#)

`unquote2(<string>,<quote-string>)` : `string` //since 4.11.5.2 [W/L] see also : [\[unquote2\]](#)

`stringReplace(<string>, <oldPattern>, <newPattern>)` : `string` //since 4.11.3 [W/L] see also : [\[stringReplace\]](#)

`strLength(<string>)` : `string (number)` //since 4.11.3 [W/L] see also : [\[strLength\]](#)

`strPos(<string>, <sub string>)` : `string (number)` //since 4.11.3 [W/L] see also : [\[strPos\]](#)

`strPart(<string>, <start pos>, <number of chars>)` : `string` //since 4.11.3 [W/L] see also : [\[strPart\]](#)

`getValue(<key string>, <hash string list>)` : `string` [W/L] see also : [\[getValue\]](#)

`getValueBySeparator(<key string>,<separator string>,<hash string list>)` : `string` //since 4.11.2.1 [W/L] see also : [\[getValueBySeparator\]](#)

`getValueFromFile(<key string>, <file name>)` : `string` //since 4.11.4.4 [W/L] see also : [\[getValueFromFile\]](#)

`getValueFromFileBySeparator(<key string>,<separator string>,<file name>)` : `string` //since 4.11.4.4 [W/L] see also : [\[getValueFromFileBySeparator\]](#)

`EscapeString: <sequence of characters>` : `string`// [W/L] see also : [\[EscapeString\]](#)

2.4.19 Stringlist handling functions [W/L]

see also : Abschnitt [9.4](#)

`getListContaining(<list>,<search string>)` : `stringlist` [W/L] see also : [\[getListContaining\]](#)

`getListContainingList(<list1>,<list2>)` : `stringlist` //since 4.11.3.7 [W/L] see also : [\[getListContainingList\]](#)

`getIndexFromListByContaining(<list> : stringlist,<search string> : string)`` : `<number> : string` //since 4.12.0.13 [W/L] see also : [\[getIndexFromListByContaining\]](#)

`count (<list>)` : `string (number)` [W/L] see also : [\[count\]](#)

`emptylist (<list>)` : `stringlist` //since 4.11.3.7 [W/L] see also : [\[emptylist\]](#)

`for %<identifier>% in <list> do <one statement | sub section>` [W/L]

```
for %s% in $list1$ do sub_test_string
```

see also : Abschnitt [9.4.10](#)

`createStringList (<string0>, <string1> ,...)` : `stringlist` [W/L]

```
set $list1$ = createStringList ('a','b')
```

see also : [\[createStringList\]](#)

`reverse (<list>)` : `stringlist` [W/L] see also : [\[reverse\]](#)

`getSubList (<start index> : <end index>, <list>)` : `stringlist` [W/L] see also : [\[getSubList\]](#)

`getSubListByMatch (<search string>, <target list>)` : `stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByMatch_sl\]](#)

`getSubListByMatch (<search list>, <target list>)` : `stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByMatch_ll\]](#)

`getSubListByContaining (<search string>, <target list>)` : `stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByContaining_sl\]](#)

`getSubListByContaining (<search list>, <target list>)` : `stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByContaining_ll\]](#)

`getSubListByKey (<search string>, <target list>)` : `stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByKey_sl\]](#)

`getSubListByKey (<search list>, <target list>)` : `stringlist` //since 4.12.0.14 [W/L] see also : [\[getSubListByKey_ll\]](#)

`getKeyList (<list>)` : `stringlist` //since 4.12.0.14 [W/L] see also : [\[getKeyList\]](#)

`addtolist(<list>,<string>)` : `stringlist` //since 4.10.8 [W/L] see also : [\[addtolist\]](#)

`addListToList(<dest list>,<src list>)` : `stringlist` //since 4.10.8 [W/L] see also : [\[addListToList\]](#)

`reencodestrlist(<list>, <from>, <to>)` : `stringlist` //since 4.11.4.2 [W/L] see also : [\[reencodestrlist\]](#)

`removeFromListByContaining(<search string>, <target list>)` : `stringlist` //since 4.11.5.1 [W/L] see also : [\[removeFromListByContaining_str\]](#)

`removeFromListByContaining(<search list>, <target list>)` : `stringlist` //since 4.11.5.1 [W/L] see also : [\[removeFromListByContaining_list\]](#)

`removeFromListByMatch(<searchstring>,<target list>)` : `stringlist` //since 4.11.6 [W/L] see also : [\[removeFromListByMatch\]](#)

`takeString (<index>, <list>)` : `string` [W/L] see also : [\[takeString\]](#)

`takeFirstStringContaining(<list>,<search string>)` : `string` [W/L] see also : [\[takeFirstStringContaining\]](#)

`setStringInListAtIndex(<newstring>,<list>,<indexstr>)` : `stringlist` //since 4.11.6 [W/L] see also : [\[setStringInListAtIndex\]](#)

[jsonAsArrayToStringList\(<jsonstr>\)](#) : stringlist //since 4.11.6: [W/L] see also : [\[jsonAsArrayToStringList\]](#)
[jsonStringListToJsonArray\(<strlist>\)](#) : jsonstr //since 4.11.6: [W/L] see also : [\[jsonStringListToJsonArray\]](#)
[jsonAsObjectGetKeyList\(<jsonstr>\)](#) : stringlist //since 4.11.6: [W/L] see also : [\[jsonAsObjectGetKeyList\]](#)
[splitString\(<string1>, <string2>\)](#) : stringlist [W/L]

```
set $list1$ = splitString ("\\server\share\dir", "\")
```

see also : [\[splitString\]](#)

[splitStringOnWhiteSpace \(<string>\)](#) : stringlist [W/L] see also : [\[splitStringOnWhiteSpace\]](#)

[composeString\(<string list>, <Link>\)](#) : string [W/L] see also : [\[composeString\]](#)

[getValue\(<key string>, <hash string list> \)](#) : string [W/L] see also : [\[getValue\]](#)

[getValueBySeparator\(<key string>, <separator string>, <hash string list> \)](#) : string //since 4.11.2.1 [W/L] see also : [\[getValueBySeparator\]](#)

2.4.20 Time / Date related functions [W/L]

[sleepSeconds <Integer> or <string>](#) : noresult [W/L]

breaks the program execution for <string> seconds. <string> has to represent an Integer Value

see also : [\[sleepSeconds\]](#)

[markTime](#) : noresult

sets a time stamp for the current system time and logs it. see also : [\[markTime\]](#)

[getDiffTimeSec](#) : string (Time in seconds since last marktime) //since 4.11.3 [W/L] see also : [\[getDiffTimeSec\]](#)

[timeStampAsFloatStr](#) : string (Floating Number - format: *days.decimal days*) //since 4.11.6 [W/L] see also : [\[timeStampAsFloatStr\]](#)

2.4.21 Usercontext / loginscripts related functions [W]:

[GetUserSID\(<Windows Username>\)](#) : string see also : [\[GetUserSID\]](#)

[GetLoggedInUser](#) : string //since 4.11.1.2 see also : [\[GetLoggedInUser\]](#)

[GetUsercontext](#) : string //since 4.11.1.2 see also : [\[GetUsercontext\]](#)

[GetScriptMode](#) : string possible values *Machine, Login* //since 4.11.2.1 see also : [\[GetScriptMode\]](#)

[saveVersionToProfile](#) : noresult - save productversion-packageversion to local profile //since 4.11.2.1 see also : [\[saveVersionToProfile\]](#)

[readVersionFromProfile](#) : string - read productversion-packageversion from local profile //since 4.11.2.1 see also : [\[readVersionFromProfile\]](#)

[scriptWasExecutedBefore](#) : boolean - is true if saved and running productversion-packageversion are identical //since 4.11.2.1 see also : [\[scriptWasExecutedBefore\]](#)

Kapitel 3

Einführung

Das Open-Source-Programm *opsi-winst* (unter Windows) bzw. *opsi-script* unter Linux fungiert im Kontext des Systems opsi – Open PC Server Integration (www.opsi.org) – als zentrale Instanz zur Abwicklung der automatischen Softwareinstallation und -Konfiguration. Es kann aber auch stand alone als Setup-Rahmen-Programm verwendet werden.

opsi-winst/opsi-script ist im Kern ein Interpreter für eine eigene, einfache Skriptsprache mit der alle, für die Software-Installation relevanten, Schritte ausgedrückt werden können.

Eine Software Installation, die auf einem *opsi-winst/opsi-script* Skript basiert, bietet verschiedene Vorteile im Vergleich zu Verfahren, die auf Kommando-Zeilen-Aufrufen beruhen (z.B. Kopieren etc.):

- *opsi-winst/opsi-script* bietet die Möglichkeit eines sehr detaillierten Protokolls der Installation. So lassen sich Fehler bei der Installation und beim Einsatz oder andere Problemsituationen frühzeitig erkennen.
- Kopieraktionen können sehr differenziert konfiguriert werden, in wie weit vorhandene Dateien überschrieben werden sollen. Dateien (z.B. DLLs) können beim Kopieren auf ihre interne Version überprüft werden.
- Ein Zugriff auf die Windows-Registry ist in unterschiedlichen Modi (z.B. vorhandene Werte überschreiben/nur neue Werte eintragen/Werte ergänzen) möglich.
- Eintragungen z.B. in die Windows-Registry können auch für alle User (einschließlich dem Default-User, der als Vorlage für die künftig angelegten User dient) vorgenommen werden.
- Es existiert eine ausgearbeitete und gut verständliche Syntax zum Patchen von XML-Konfigurationsdateien, das in die sonstigen Konfigurationsaufgaben integriert ist.

Kapitel 4

Betrieb unter Linux (*opsi-script*)

4.1 Einführung

Seit Version 4.11.4 steht der *opsi-winst* unter dem Namen *opsi-script* auch unter Linux zur Verfügung.

Bedingt durch den Fortgang der Portierung und durch die Unterschiede der Betriebssysteme stehen bestimmte Funktionalitäten nicht unter beiden Betriebssystemen zur Verfügung. Im folgenden sind entsprechende Abschnitte markiert:

- [W/L] sowohl unter Windows als auch unter Linux verfügbar
- [W] nur unter Windows verfügbar
- [L] nur unter Linux verfügbar

4.2 Wichtige Unterschiede und Hinweise

opsi-script ist im Moment ein GUI Programm, welches ohne Zugriff auf ein X Display nicht gestartet werden kann.

opsi-script-nogui ist eine Kommandozeilen Version, welche sich auch ohne X starten lässt.

Unter Linux ist das Parameterzeichen nicht "/" sondern "-". Also statt unter Windows `opsi-winst /help` unter Linux `opsi-script -help`.

4.3 Pfade

Unter Linux sind die unterschiedlichen Komponenten gemäß des Linux Filesystem Hierarchie Standard nicht an einer Stelle zu finden. Daher hier ein Überblick:

- Ausführbare Programme:
`/usr/bin/opsi-script`
`/usr/bin/opsi-script-nogui`
- Verzeichnisse für Logdateien:
Ausgeführt als root: `/var/log/opsi-script`
Ausgeführt als user: `/tmp`
- Language files:
`/usr/share/locale`

- Skin files:
Default = /usr/share/opsi-script/skin
Custom = /usr/share/opsi-script/customskin
- opsi-script library files:
/usr/share/opsi-script/lib
- Config files:
/etc/opsi-script
- Variable files:
/var/lib/opsi-client-agent/opsi-script

4.4 Verwendung von Pfaden in opsiscript

Seit Version 4.11.4 wird bei allen Funktionen welche einen Pfad erwarten, intern eine Funktion aufgerufen welche den übergebenen String in einen für das Betriebssystem gültigen Pfad wandeln. Es werden also alle Pfadtrennzeichen korrekt gesetzt. So wird z.B. aus dem Pfad /home/opsiproduct\myproduct\CLIENT_DATA unter Linux /home/opsiproduct/myproduct/CLIENT_DATA. Dies bedeutet auch, dass unter Linux keine Dateien angelegt oder verarbeitet werden können welche einen Backslash im Namen haben.

4.5 Linuxspezifische Funktionen

Zur Unterstützung von Linux gibt es folgende Linuxspezifische Funktionen:

- GetOS // *Linux* or *Windows_NT* [W/L]
- getLinuxDistroType // *debian* or *redhat* or *suse* [L]
- getLinuxVersionMap [L]
- chmod in Files Sektionen [L]

4.6 Beispiel Scripte für Linux

4.6.1 Run on Linux only

```
[Actions]
DefVar $OS$

set $OS$ = GetOS

if not ($OS$ = "Linux")
    logError "Installation aborted: wrong OS version: only Linux allowed"
    isFatalError "wrong OS"
endif
```

4.6.2 Which Linux Version

```
[Actions]
DefVar $distCodeName$
DefVar $distroName$
DefVar $distRelease$
DefVar $distrotype$

DefStringList $linuxInfo$

set $distrotype$ = getLinuxDistroType
set $linuxInfo$ = getLinuxVersionMap
set $distCodeName$ = getValue("Codename", $linuxInfo$)
set $distRelease$ = getValue("Release", $linuxInfo$)
set $distroName$ = getValue("Distributor ID", $linuxInfo$)
```

Tabelle 4.1: getLinuxVersionMap Result Examples

Distro	Distributor ID	Release	Codename	Description
Ubuntu Lucid	Ubuntu	10.04	lucid	
Ubuntu Precise	Ubuntu	12.04	precise	Ubuntu 12.04.5 LTS
Ubuntu Trusty	Ubuntu	14.04	trusty	
Ubuntu Xenial	Ubuntu	16.04	xenial	
Debian 6	Debian	6.0.10	squeeze	Debian GNU/Linux 6.0.10 (squeeze)
Debian 7	Debian	7.6	wheezy	Debian GNU/Linux 7.6 (wheezy)
Debian 8	Debian	8.3	jessie	Debian GNU/Linux 8.3 (jessie)
openSUSE 12.3	openSUSE project	12.3	Dartmouth	openSUSE 12.3 (x86_64)
openSUSE 13.1	openSUSE project	13.1	Bottle	openSUSE 13.1 (Bottle) (x86_64)
openSUSE 13.2	openSUSE project	13.2	Bottle	openSUSE 13.2 (Bottle) (x86_64)
openSUSE Leap 42.1	SUSE LINUX	42.1	n/a	openSUSE Leap 42.1 (x86_64)
SLES11SP3	SUSE LINUX	11	n/a	SUSE Linux Enterprise Server 11 (x86_64)
SLES12	SUSE LINUX	12	n/a	SUSE Linux Enterprise Server 12
SLES12SP1	SUSE LINUX	12.1	n/a	SUSE Linux Enterprise Server 12 SP1
Fedora20	Fedora	20		
CentOS 6.5	CentOS	6.5	Final	
CentOS 7.0	CentOS	7.0.1406	Core	CentOS Linux release 7.0.1406 (Core)
RedHat 6.5	RedHatEnterpriseServer	6.5	Santiago	Red Hat Enterprise Linux Server release 6.5 (Santiago)

Tabelle 4.1: (continued)

Distro	Distributor ID	Release	Codename	Description
RedHat 7.0	RedHatEnterpriseServer	7.0	Maipo	Red Hat Enterprise Linux Server release 7.0 (Maipo)
UCS 3.2	Univention	3.2-3 errata221	Borgfeld	Univention Corporate Server 3.2-3 errata221 (Borgfeld)
UCS 4.0	Univention	????	???	????
UCS 4.1	Univention	4.1-1 errata122	Vahr	Univention Corporate Server 4.1-1 errata122 (Vahr)

4.6.3 ShellInAnIcon call

```
[Actions]

ShellInAnIcon_ls

[ShellInAnIcon_ls]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
ls
exit $?
```

It is always a good idea to start with `set -x` for more information in the log and to set the PATH. You should end with `exit $?` so that the exitcode of the last call is the exitcode of the section.

4.6.4 Add a repository

Ubuntu / Debian

```
[Actions]
DefVar $newrepo$

set $newrepo$ = "deb http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40/Debian_7
.O/ ./"

comment "Method 1: use add-apt-repository ..."
ShellInAnIcon_add_rep_deb
ShellInAnIcon_add_repokey_deb
comment "Method 2: use add-apt-repository ..."
PatchTextFile_add_repo_deb "/etc/apt/sources.list"
ShellInAnIcon_add_repokey_deb

[ShellInAnIcon_add_rep_deb]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
export DEBIAN_FRONTEND=noninteractive
apt-get --yes --force-yes install software-properties-common
```

```

apt-get --yes --force-yes install python-software-properties
add-apt-repository '$newrepo$'
exit $?

[PatchTextFile_add_repo_deb]
FindLine_StartingWith "$newrepo$"
DeleteTheLine
GoToBottom
InsertLine "$newrepo$"

[ShellInAnIcon_add_repokey_deb]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
wget --no-check-certificate -O - $newrepo$/Release.key | apt-key add -
apt-get update
exit $?

```

SUSE

```

[Actions]
DefVar $newrepo$

set $newrepo$ = "http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40/opensUSE_13
    .1/home:uibmz:opsi:opsi40.repo"

ShellInAnIcon_add_opsi_repository_suse

[ShellInAnIcon_add_opsi_repository_suse]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
zypper --no-gpg-checks --non-interactive --gpg-auto-import-keys ar --refresh $newrepo$
zypper --no-gpg-checks --non-interactive --gpg-auto-import-keys refresh
exit $?

```

CentOS / Redhat

```

[Actions]
DefVar $newrepo$

set $newrepo$ = "http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40/CentOS_7/
    home:uibmz:opsi:opsi40.repo"

comment "Method 1: use wget ..."
ShellInAnIcon_add_repo_redhat
ShellInAnIcon_refresh_repo_redhat
comment "Method 2: use PatchTextFile ..."
PatchTextFile_add_repo_redhat "/etc/yum.repos.d/mynew.repo"
ShellInAnIcon_refresh_repo_redhat

ShellInAnIcon_add_repo_redhat

[ShellInAnIcon_add_repo_redhat]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
yum -y install wget
cd /etc/yum.repos.d
wget --no-check-certificate $newrepo$

```

```

exit $?

[PatchTextFile_add_repo_redhat]
AppendLine "[home_uibmz_opsi_opsi40]"
AppendLine "name=opsi 4.0 (CentOS_7)"
AppendLine "type=rpm-md"
AppendLine "baseurl=http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40/CentOS_7/"
AppendLine "gpgcheck=1"
AppendLine "gpgkey=http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40/CentOS_7/"
AppendLine "repodata/repomd.xml.key"
AppendLine "enabled=1"

[ShellInAnIcon_refresh_repo_redhat]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
yum makecache
yum -y repolist
exit $?

```

4.6.5 Delete a repository

Ubuntu / Debian

```

[Actions]
DefVar $delrepo$
DefStringlist = $resultlist$

set $delrepo$ = "deb http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40/Debian_7
               .0/ ./"

if LineBeginning_ExistsIn($delrepo$, "/etc/apt/sources.list")
    PatchTextFile_del_repo_deb "/etc/apt/sources.list"
    set $resultlist$ = shellCall("apt-get update")
endif

[PatchTextFile_del_repo_deb]
FindLine_StartingWith "$delrepo$"
DeleteTheLine

```

SUSE

```

[Actions]
DefVar $delrepo$

comment "$delrepo$ is the section name of the repo file in /etc/zypp/repos.d/"
comment "$delrepo$ can be found by zypper lr"
set $delrepo$ = "home_uibmz_opsi_opsi40"
ShellInAnIcon_del_opsi_repository_suse

[ShellInAnIcon_del_opsi_repository_suse]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
zypper --non-interactive rr $delrepo$
exit $?

```

CentOS / Redhat

```
[Actions]
DefVar $delrepo$

comment "$delrepo$ ist the name of the repo file in /etc/yum.repos.d"
set $delrepo$ = "/etc/yum.repos.d/home:uibmz:opsi:opsi40.repo"

[ShellInAnIcon_del_opsi_repository_redhat]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
rm $delrepo$
yum makecache
yum -y repolist
exit $?
```

4.6.6 Installing a package**Ubuntu / Debian**

```
[Actions]

ShellInAnIcon_install_wget_debian

[ShellInAnIcon_install_wget_debian]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
export DEBIAN_FRONTEND=noninteractive
apt-get --yes --force-yes install wget
exit $?
```

The DEBIAN_FRONTEND=noninteractive and the apt-get options --yes --force-yes are needed for fully non interactive install.

SUSE

```
[Actions]

ShellInAnIcon_install_wget_suse

[ShellInAnIcon_install_wget_suse]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
zypper --no-gpg-checks --non-interactive install wget
exit $?
```

The zypper options --no-gpg-checks --non-interactive are needed for fully non interactive install.

CentOS / Redhat

```
[Actions]

ShellInAnIcon_install_wget_redhat

[ShellInAnIcon_install_wget_redhat]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
yum -y install wget
exit $?
```

The yum option `-y` is needed for fully non interactive install.

4.6.7 Integrated Example

```
[Actions]

DefVar $OS$
DefVar $distro_type$

DefStringlist $list$

set $OS$ = GetOS

if not ($OS$ = "Linux")
    logError "Installation aborted: wrong OS version: only Linux allowed"
    isFatalError "wrong OS"
endif

set $distro_type$ = getLinuxDistroType
set $list$ = getLinuxVersionMap

comment "install wget ..."
if $distro_type$ = "redhat"
    ShellInAnIcon_install_wget_redhat
else
    if $distro_type$ = "suse"
        ShellInAnIcon_install_wget_suse
    else
        if $distro_type$ = "debian"
            ShellInAnIcon_install_wget_debian
        else
            LogError "Unknown distro type"
        endif
    endif
endif
endif

[ShellInAnIcon_install_wget_debian]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
export DEBIAN_FRONTEND=noninteractive
apt-get --yes --force-yes install wget
exit $?

[ShellInAnIcon_install_wget_redhat]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
yum -y install wget
exit $?

[ShellInAnIcon_install_wget_suse]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
zypper --no-gpg-checks --non-interactive install wget
exit $?
```

Kapitel 5

Start und Aufrufparameter

Der *opsi-winst/opsi-script* enthält unter Windows seit Version 4.11.3 ein Manifest mit der Option: `<requestedExecutionLevel level="highestAvailable" />`. Dies bedeutet, dass unter NT6 als Administrator aufgerufen, versucht wird als *elevated* Prozess zu arbeiten. Wird der *opsi-winst/opsi-script* mit User Rechten aufgerufen, so läuft er unter den Rechten dieses Users.

Wird der *opsi-winst/opsi-script* ohne Parameter aufgerufen, so startet er interaktiv.

opsi-winst/opsi-script kann ja nach Kontext und Verwendungszweck mit unterschiedlichen Parametern gestartet werden.

Es existieren folgende Syntax-Varianten des Aufrufs:

(1) Anzeigen der Varianten:

```
opsi-winst /?
opsi-winst /h[elp]
```

(2) Ausführung eines Skripts:

```
opsi-winst <scriptfile> [/logfile <logfile> ]
[/batch | /histolist <opsi-winstconfigfilepath>]
[/productid <productid> ]
[/usercontext <[domain\]username> ]
[/parameter parameterstring]
```

(3) Ausführen einer Liste von Skripts:

```
opsi-winst /scriptfile <scriptfile> [;<scriptfile>]*
[ /logfile <logfile> ]
[/batch | /silent ]
[/usercontext <[domain\]username> ]
[/parameter <parameterstring>]
```

(4) Abarbeitung des Software-Profiles über den opsi Service mit der entsprechenden Umsetzung (seit *opsi-winst/opsi-script* 4.11.2)

```
opsi-winst /opsservice <opsserviceurl>
/clientid <clientname>
/username <username>
/password <password>
/sessionid <sessionid>
[/usercontext <[domain\]username>]
[/allloginscripts | /loginscripts | /productlist <productid>[,<productid>]* ]
[/silent]
```

Note

Unter Linux ist das Parameterzeichen nicht "/" sondern "-". Also statt unter Windows `opsi-winst /help` unter Linux `opsi-script -help`.

Generelle Erläuterungen:

- Default Name für die Logdatei ist unter Windows (ab 4.11.4) `C:\opsi.org\log\opsi-script.txt`
- Der Parameterstring, angekündigt durch die Option `/parameter`, wird an das jeweils aufgerufene *opsi-winst/opsi-script* Skript (über die String-Funktion `ParamStr`) übergeben.

Erläuterungen zu (2) und (3):

- Die Anwendung der Option `/batch` bewirkt, dass nur die Batch-Oberfläche angezeigt wird, die keine Möglichkeiten für Benutzereingaben bietet. Bei der Option `/silent` wird die Batch-Oberfläche ausgeblendet. Beim Aufruf ohne den Parameter `/batch` erscheint die Dialog-Oberfläche. Mit ihr ist die interaktive Auswahl von Skript- und Protokolldatei möglich (in erster Linie für Testzwecke gedacht).
- Wenn der Aufruf mit der Option `/usercontext <[domain]username>` erfolgt, kann die Konfiguration für einen spezifizierten eingeloggten Nutzer erfolgen (besonders im Zusammenhang mit der *User Profile Management* Erweiterung).
- Die Verwendung des Parameters `/histolist` gefolgt von `opsi-winstconfigfilepath` bewirkt, dass in der Dialog-Oberfläche das Eingabefeld für den Skript-Dateinamen mit einer Historienliste erscheint und automatisch die zuletzt verwendete Datei erneut vorgeschlagen wird. Wenn `opsi-winstconfigfilepath` nur ein Verzeichnis benennt (mit "\" abgeschlossen), wird als Dateiname `WINST.INI` verwendet.

Erläuterungen zu (4):

- Default für `clientid` ist der full qualified Computername.
- Die Option `/allloginscripts` oder `/loginscripts` schaltet das Verhalten auf das Abarbeiten von *userLoginScripts* um.
Siehe hierzu im opsi-manual das Kapitel *User Profile Management*.
- Die Option `/silent` schaltet die Batchoberfläche ab (keine Ausgaben).

Die Skripte werden per default im Batchmodus abgearbeitet.

5.1 Logging Dateien und Pfade

Die Default Logdatei heist `opsi-script.log`. Es werden bis zu 8 Sicherungskopien angelegt: `opsi-script_0.log` bis `opsi-script_8.log`.

Die Logdateien werden im Encoding UTF-8 angelegt.

Die default Protokolldateien werden unter Windows in das Verzeichnis `c:\opsi.org\log` geschrieben, welches der *opsi-winst/opsi-script* zu erstellen versucht. Wenn der *opsi-winst/opsi-script* nicht erfolgreich bei der Erstellung dieses Protokollverzeichnisses ist, wird das Benutzer TEMP-Verzeichnis zum Speichern der Protokolldatei genutzt.

Logdateien unter Linux: Ausgeführt als root (default): `/var/log/opsi-script` Ausgeführt als user: `/tmp`

Der Name der Protokolldatei und der Speicherort können durch eine spezifizierte Kommandozeile überschrieben werden.

In dem Fall, dass der *opsi-winst/opsi-script* ein Skript im `/batch` mode und mit einem spezifizierten (und funktionierenden) User Kontext aufgerufen wird, ist der voreingestellte Protokollpfad `opsi/tmp` in dem Anwendungsverzeichnis des Benutzers. Dieses wird überschrieben, wenn ein anderer Protokollpfad angegeben ist.

5.2 Zentrale Konfiguration über opsi Configs (Host Parameter)

Über opsi Configs (**Host-Parameter**) kann das Logging beeinflusst werden:

- `opsi-script.global.debug_prog` : boolean
Wenn false werden Logmeldungen welche zum Debuggen des opsi-script selber dienen nicht ausgegeben, soweit es sich nicht um Warnungen oder Fehler handelt.
Default: false
Damit werden die Logdateien entlastet und nur noch Meldungen die Script relevant sind, stehen in den Logdateien. Die Umstellung der entsprechenden Logmeldungen im Quellcode des opsi-script, ist noch nicht abgeschlossen und wird bei ca. 1700 Log aufrufen auch noch etwas dauern.
- `opsi-script.global.debug_lib` : boolean
Wenn false, so werden Logmeldungen aus lokalen Funktionen welche aus Libraries importiert wurden, nur ausgegeben soweit es sich um Warnungen oder Fehler handelt.
Default : false
- `opsi-script.global.default_loglevel` : intstr
Setzt (überschreibt) den Standard defaultloglevel von opsi-script.
Dieser Config hat keinen Einfluss auf Scripte bei denen der Loglevel per `setLogLevel` explizit gesetzt worden ist.
Default : 7
siehe auch [\[SetLogLevel\]](#)
- `opsi-script.global.force_min_loglevel` : intstr
Erzwingt einen minimalen Loglevel.
Dies dient dazu bei der Entwicklung und/oder Fehlersuche gezielt und temporär für einzelne Clients den Loglevel zu erhöhen ohne hierzu Anpassungen am Script vornehmen zu müssen.
Default: 0
siehe auch [\[SetLogLevel\]](#)
- `opsi-script.global.ScriptErrorMessages` : boolean
Wenn false werden Syntax-Fehlermeldungen nicht interaktiv ausgegeben sondern nur in die Logdatei geschrieben. Im Produktivbetrieb ist es sinnvoll das dieser Parameter false ist. Daher ist Default für diesen Config=false. Der Default von opsi-script für diesen Parameter ist (aus historischen Gründen) true. Im Servicekontext überschreibt der Config den Default von opsi-script. Ausserhalb des Servicekontext gilt der Default von opsi-script. Diese Default Werte können innerhalb eines scriptes mit der Anweisung `ScriptErrorMessages` überschrieben werden. Default: false
see also : [\[ScriptErrorMessages\]](#)
- `opsi-script.global.AutoActivityDisplay` : boolean
Wenn true wird während des Laufs von externen Prozessen (winbatch,dosbatch,execwith Sektionen) ein (marquee) Fortschrittsbalken (der Endlos durch läuft) angezeigt.
Default: true
see also : [\[AutoActivityDisplay\]](#)

Kapitel 6

Weitere Konfigurationsoptionen

6.1 Zentrales Protokollieren von Fehlermeldungen

opsi-winst kann die Protokoll-Informationen über den opsi-web-service an den opsi-server senden.

6.2 Skinnable *opsi-winst/opsi-script* [W/L]

Note

Für Linux siehe: Abschnitt [4.3](#)

Ab Version 3.6 verfügt *opsi-winst/opsi-script* einen veränderbare Oberfläche. Seine Elemente liegen im Unterverzeichnis `winstskin` des Verzeichnisses, in dem der ausgeführte *opsi-winst/opsi-script* liegt. Die editierbare Definitionsdatei ist `skin.ini`.

Seit Version 4.11.3.5 sucht der *opsi-winst* nach dem zu verwendenden Skin Verzeichnis in folgender Reihenfolge, wobei das erste Verzeichnis, welches eine `skin.ini` enthält verwendet wird:

1. `%WinstDir%\..\custom\winstskin`
2. `%WinstDir%\winstskin`

Mit dem Befehl `SetSkinDirectory` kann ein `SkinDirectory` auch im Script angegeben werden. Wird bei diesem Befehl ein leerer oder ungültiger Pfad angegeben, so wird der Defaultpfad verwendet.

Beispiel:

```
SetSkinDirectory "%ScriptPath%\testskin"  
sleepseconds 1  
SetSkinDirectory ""
```

6.3 *opsi-winst/opsi-script* encoding [W/L]

Das Default Encoding für ein Script ist das Encoding des Systems auf dem der *opsi-winst/opsi-script* läuft. D.h. auf einem Griechischen System wird das script mit unter Windows mit cp1253 interpretiert während das selbe Script auf einem deutschem Windows System mit cp1252 und auf einem Linux System mit UTF-8 interpretiert wird.

Seit Version 4.11.4.1 kann bei einem Script (egal ob Hauptsript, sub oder include) das encoding auch angegeben werden. Dazu gibt es den Befehl:

```
encoding=<encoding>
```

Dieser Befehl kann an einer beliebigen Stelle in der Scriptdatei stehen. Das `<encoding>` ist eines der folgenden Werte:

Tabelle 6.1: Encodings

encoding	erlaubter alias	Bemerkung
system		verwende encoding des laufenden OS
auto		versuche das encoding zu erraten.
UTF-8	utf8	
UTF-8BOM	utf8bom	
Ansi	ansi	cp1252/ISO 8859-1
CP1250	cp1250	Zentral- und osteuropäische Sprachen
CP1251	cp1251	Kyrillisches Alphabet
CP1252	cp1252	Westeuropäische Sprachen
CP1253	cp1253	Griechisches Alphabet
CP1254	cp1254	Türkisches Alphabet
CP1255	cp1255	Hebräisches Alphabet
CP1256	cp1256	Arabisches Alphabet
CP1257	cp1257	Baltische Sprachen
CP1258	cp1258	Vietnamesische Sprachen
CP437	cp437	Die ursprüngliche Zeichensatztable des IBM-PC
CP850	cp850	"Multilingual (DOS-Latin-1)", westeuropäische Sprachen
CP852	cp852	Slawische Sprachen (Latin-2), zentraleuropäische und osteuropäische Sprachen
CP866	cp866	Kyrillisches Alphabet
CP874	cp874	Thai Alphabet
CP932	cp932	Japanische Schreibsysteme (DBCS)
CP936	cp936	GBK für chinesische Kurzzeichen (DBCS)
CP949	cp949	Hangul/Koreanische Schriftzeichen (DBCS)
CP950	cp950	Chinesische Langzeichen (DBCS)
ISO-8859-1	iso8859-1	Latin-1
ISO-8859-2	iso8859-2	Latin-2
KOI-8	koi8	Kyrillisches Alphabet
UCS-2LE	ucs2le	(UTF-16-LE)
UCS-2BE	ucs2be	(UTF-16-BE, Windows NT Standard)

Quellen auch:

<https://de.wikipedia.org/wiki/Codepage>

<http://msdn.microsoft.com/en-us/library/windows/desktop/dd317752%28v=vs.85%29.aspx>

<http://msdn.microsoft.com/en-us/library/cc195054.aspx>

<https://de.wikipedia.org/wiki/Ansi>

<https://de.wikipedia.org/wiki/UTF-8>

Kapitel 7

Das *opsi-winst/opsi-script* Skript

Wie schon erwähnt, interpretiert das Programm *opsi-winst/opsi-script* eine eigene, einfache Skriptsprache, die speziell auf die Anforderungen von Softwareinstallationen zugeschnitten ist. Jede Installation wird durch ein spezifisches Skript beschrieben und gesteuert.

In diesem Abschnitt ist der Aufbau eines *opsi-winst/opsi-script* Skripts im Überblick skizziert – etwa so, wie man es braucht, um die Funktionsweise eines Skripts in etwas nachvollziehen zu können.

Sämtliche Elemente werden in den nachfolgenden Abschnitten genauer beschrieben, so dass auch deutlich wird, wie Skripte entwickelt und abgeändert werden können.

7.1 Ein Beispiel

In ihrer äußeren Form ähneln die *opsi-winst/opsi-script* Skripte .INI-Dateien. Sie setzen sich aus einzelnen Abschnitten (Sektionen) zusammen, die jeweils durch eine Überschrift (den Sektionsnamen) in eckigen Klammern [] gekennzeichnet sind. Ein beispielhaftes, schematisches *opsi-winst/opsi-script* Skript (hier mit einer Fallunterscheidung für verschiedene Betriebssystem-Varianten) könnte etwas so aussehen:

```
[Actions]
Message "Installation von Mozilla"
SetLogLevel=6

;Welche Windows-Version?
DefVar $MSVersion$

Set $MSVersion$ = GetMsVersionInfo
if CompareDotSeparatedNumbers($MSVersion$, ">=", "6")
    sub_install_win7
else
    if ( $MSVersion$ = "5.1" )
        sub_install_winXP
    else
        stop "not a supported OS-Version"
    endif
endif

[sub_install_win7]
Files_Kopieren_win7
WinBatch_Setup
```

```
[sub_install_winXP]
Files_Kopieren_XP
WinBatch_SetupXP

[Files_Kopieren_win7]
copy "%scriptpath%\files_win7\*.*" "c:\temp\installation"

[Files_Kopieren_winxp]
copy "%scriptpath%\files_winxp\*.*" "c:\temp\installation"

[WinBatch_Setup]
c:\temp\installation\setup.exe

[WinBatch_SetupXP]
c:\temp\installation\install.exe
```

Wie lassen sich die Sektionen oder Abschnitte dieses Skripts lesen?

7.2 Primäre und sekundäre Unterprogramme des *opsi-winst/opsi-script* Skripts

Das das Skript insgesamt als die Vorschrift zur Ausführung einer Installation anzusehen ist, d.h. als eine Art von Programm, kann jeder seiner Sektionen als Teil- oder Unterprogramm (auch als "Prozedur" oder "Methode" bezeichnet) aufgefasst werden.

Das Skript besteht demnach aus einer Sammlung von Teilprogrammen. Damit ein Mensch oder ein Interpreter-Programm es richtig liest, muss bekannt sein, welches der Teilprogramme Priorität haben, mit welchem also in der Abarbeitung angefangen werden soll.

Für die *opsi-winst/opsi-script* Skripte ist festgelegt, dass die primäre Sektion mit dem Titel [Actions] abgearbeitet wird. Alle anderen Sektionen fungieren als Unterprogramme und können in diesen beiden Sektionen aufgerufen werden. Nur in den Sub-Sektionen können dann wiederum Unterprogramme aufgerufen werden.

Anmerkung

Wird ein Script als *userLoginScript* aufgerufen, und enthält eine Sektion [ProfileActions] so wird das Script ab dieser Sektion abgearbeitet.

Dies liefert die Grundlage für die Unterscheidung zwischen primären und sekundären Unterprogrammen:

Die primären oder Steuerungssektionen umfassen:

- die **Actions**-Sektion
- die **Sub**-Sektionen (Unterprogramme der Actions-Sektion, die auch deren Syntax und Funktionslogik erben).
- die **ProfileActions**-Sektion die je nach script mode (Machine/Login) unterschiedlich interpretiert wird.

In diesen Sektionsarten können andere Sektionstypen aufgerufen werden, so dass der Ablauf des Skripts "im Großen" geregelt wird.

Dagegen weisen die sekundären, aufgabenspezifischen Sektionen eine eng an die jeweilige Funktion gebundene Syntax auf, die keinen Verweis auf andere Sektionen erlaubt. Derzeit existieren die folgenden Typen sekundärer Sektionen:

- Files-Sektionen,
- WinBatch-Sektionen,

- DosBatch/DosInAnIcon/ShellInAnIcon-Sektionen,
- Registry-Sektionen
- Patches-Sektionen,
- PatchHosts-Sektionen,
- PatchTextFile-Sektionen,
- XMLPatch-Sektionen,
- LinkFolder-Sektionen,
- opsiServiceCall-Sektionen,
- ExecPython-Sektionen,
- ExecWith-Sektionen,
- LDAPsearch-Sektionen.

Im Detail wird Bedeutung und Syntax der unterschiedlichen Sektionstypen in den Abschnitten Kapitel 9 und Kapitel 10 behandelt.

7.3 String-Ausdrücke im *opsi-winst/opsi-script* Skript

In den primären Sektionen können textuelle Angaben (String-Werte) auf verschiedene Weisen bestimmt werden:

- Durch die direkte Nennung des Inhalts, in der Regel in (doppelten) Anführungszeichen, Beispiele:
"Installation von Mozilla"
"n:\home\user name"
- Durch die Anführung einer String-Variable oder String-Konstante, die einen Wert "enthält" oder "trägt":
\$MsVersion\$
 kann – sofern der Variable zuvor ein entsprechender Wert zugewiesen wurde - für "6.1" stehen .
- Durch Aufruf einer Funktion, die einen String-Wert ermittelt:
EnvVar ("Username")
 holt z.B. einen Wert aus der Systemumgebung, in diesem Fall den Wert der Umgebungsvariable Username. Funktionen können auch parameterlos sein, z.B.
GetMsVersionInfo
 Dies liefert auf einem Win7-System wieder den Wert "6.1" (anders als bei einer Variablen wird der Wert aber bei jeder Verwendung des Funktionsnamens neu bestimmt).

Durch einen additiven Ausdruck, der einfache String-Werte bzw. -Ausdrücke zu einem längeren String verkettet (wer unbedingt will, kann dies als Anwendung der Plus-Funktion auf zwei Parameter ansehen ...).

\$Home\$ + "\mail"

(Mehr zu diesem Thema in Kapitel Abschnitt 9.3).

In den sekundären Sektionen gilt die jeweils spezifische Syntax, die z.B. beim Kopieren weitgehend der des "normalen" DOS-copy-Befehls entspricht. Daher können dort keine beliebigen String-Ausdrücke verwendet werden. Zum "Transport" von String-Werten aus den primären in die sekundären Sektionen eignen sich ausschließlich einfache Werte-Träger, also die Variablen und Konstanten.

Im nächsten Kapitel folgt genaueres zu Definition und Verwendung von Variablen und Konstanten.

Kapitel 8

Definition und Verwendung von Variablen und Konstanten im *opsi-winst/opsi-script* Skript

8.1 Allgemeines

Variable und Konstanten erscheinen im Skript als "Wörter", die vom *opsi-winst/opsi-script* interpretiert werden und Werte "tragen". "Wörter" sind dabei Zeichenfolgen, die Buchstaben, Ziffern und die meisten Sonderzeichen (insbesondere ".", "-", "_", "\$", "%"), aber keine Leerzeichen, Klammern oder Operatorzeichen ("+") enthalten dürfen.

Groß- und Kleinbuchstaben gelten als gleichbedeutend.

Es existieren folgende Arten von Werteträgern:

- Globale Text-Konstanten
enthalten Werte, die *opsi-winst/opsi-script* automatisch ermittelt und die nicht geändert werden können. Vor der Abarbeitung eines Skripts werden ihre Bezeichnungen im gesamten Skript gegen ihren Wert ausgetauscht. Die Konstante `%ScriptPath%` ist die definierte Pfad-Variable, die den Pfad angibt in dem der *opsi-winst/opsi-script* das Skript findet und ausführt. Dies könnte beispielsweise `p:\product` sein. Man müsste dann `"%ScriptPath%"` in das Skript schreiben, wenn man den Wert `p:\product` bekommen möchte.
Zu beachten sind die Anführungszeichen um die Konstantenbezeichnung.
- Text-Variable oder String-Variable
entsprechen den gebräuchlichen Variablen in anderen Programmiersprachen. Die Variablen müssen vor ihrer Verwendung mit `DefVar` deklariert werden. In einer primären Sektion kann einer Variable mehrfach ein Wert zugewiesen werden und mit den Werten in der üblichen Weise gearbeitet werden („Addieren“ von Strings, spezielle String-Funktionen).
In sekundären Sektionen erscheinen sie dagegen als statische Größen. Ihr jeweils aktueller Wert wird bei der Abarbeitung der Sektion für ihre Bezeichnung eingesetzt (so wie es bei Textkonstanten im ganzen Skript geschieht).
- Variablen für String-Listen
werden mit `DefStringList` deklariert. Eine String-Listenvariable kann ihren Inhalt, also eine Liste von Strings, auf unterschiedlichste Weisen erhalten. Mit String-Listenfunktionen können die Listen in andere Listen überführt oder als Quelle für Einzelstrings verwendet werden.

Im einzelnen:

8.2 Globale Textkonstanten [W/L]

Damit Skripte ohne manuelle Änderungen in verschiedenen Umgebungen eingesetzt werden können, ist es erforderlich, sie durch gewisse Systemeigenschaften zu parametrisieren. opsi-winst kennt einige System-Größen, die innerhalb des Skriptes als Text-Konstanten anzusehen sind.

8.2.1 Verwendung

Wichtigste Eigenschaft der Text- oder String-Konstanten ist die spezifische Art, wie die von ihnen repräsentierten Werte eingesetzt werden:

Vor Abarbeitung des Skriptes werden die Namen der Konstanten in der gesamten Skriptdatei gegen die Zeichenfolge ihrer vom *opsi-winst/opsi-script* bestimmten Werte ausgetauscht.

Diese Ersetzung vollzieht sich – in der gleichen Weise wie bei den Text-Variablen in den sekundären Sektionen – als ein einfaches Suchen- und Ersetzen-Verfahren (Search und Replace), ohne Rücksicht auf den jeweiligen Ort, an dem die Konstante steht.

8.2.2 Beispiel

opsi-winst/opsi-script kennt z.B. die Konstanten `%ScriptPath%` für den Ort im Verzeichnisbaum, an dem das interpretierte Skript steht und `%System%` für den Namen des Windows-Systemverzeichnisses. In einer `Files`-Sektion könnten daher auf folgende Weise alle Dateien eines Verzeichnissystems, das im gleichen Verzeichnis wie das Skript liegt, in das Windows-Systemverzeichnis kopiert werden:

```
[files_do_my_copying]
copy "%ScriptPath%\system\*.*" "%System%"
```

Gegenwärtig sind folgende Konstanten definiert:

8.2.3 Systemverzeichnis

8.2.3.1 Basissystemverzeichnis [W]

```
%ProgramFilesDir%: c:\program files
%ProgramFiles32Dir%: c:\Program Files (x86)
%ProgramFiles64Dir%: c:\program files
%ProgramFilesSysnativeDir%: c:\program files
%Systemroot%: c:\windows
%System%: c:\windows\system32
%Systemdrive%: c:
%ProfileDir%: c:\Documents and Settings
```

8.2.3.2 Gemeinsames (AllUsers) Verzeichnis [W]

```
%AllUsersProfileDir% or %CommonProfileDir%: c:\Documents and Settings\All Users
%CommonStartMenuPath% or %CommonStartmenuDir%: c:\Documents and Settings\All Users\Startmenu
%CommonAppdataDir%: c:\Documents and Settings\All Users\Application Data
%CommonDesktopDir%
%CommonStartupDir%
%CommonProgramsDir%
```

Constant	XP en	Win7
%AllUsersProfileDir%	c:\Documents and Settings\All Users	C:\Users\Public
%CommonProfileDir%	c:\Documents and Settings\All Users	C:\Users\Public
%CommonStartMenuPath%	c:\Documents and Settings\All Users\start menue	C:\ProgramData\Microsoft\Windows\Start Menu
%CommonAppDataDir%	c:\Documents and Settings\All Users	C:\ProgramData
%CommonDesktopDir%	c:\Documents and Settings\All Users	C:\Users\Public\Desktop
%CommonStartupDir%	c:\Documents and Settings\All Users	C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup
%CommonProgramsDir%	c:\Documents and Settings\All Users	C:\ProgramData\Microsoft\Windows\Start Menu\Programs
%AllUsersProfileDir%	c:\Documents and Settings\All Users	C:\Users\Public
%DefaultUserProfileDir%	C:\Dokumente und Einstellungen\Default User	C:\Users\Default
%ProfileDir%	c:\Documents and Settings\All Users	C:\Users
%Systemroot%	C:\Windows	C:\Windows
%System%	C:\Windows\system32	C:\Windows\system32

8.2.3.3 Default User Verzeichnis [W]

%DefaultUserProfileDir%

8.2.3.4 Aktuelles (logged in oder usercontext) User Verzeichnis [W]

%AppdataDir% or %CurrentAppdataDir% : //since 4.10.8.13

NT5: c:\Documents and Settings\%USERNAME%\Application Data NT6: c:\users\%USERNAME%\Appdata\Roaming

%CurrentStartmenuDir%

%CurrentDesktopDir%

%CurrentStartupDir%

%CurrentProgramsDir%

%CurrentSendToDir%

%CurrentProfileDir% //since 4.11.2.1

8.2.3.5 /AllNtUserProfiles Verzeichnis Konstanten [W]

%UserProfileDir%

Diese Konstante wird nur innerhalb von *Files*-Sektionen, die mit der Option */AllNtUserProfiles* aufgerufen werden, interpretiert. Sie wird dann der Reihe nach belegt mit dem Namen des Profil-Verzeichnisses der, verschiedenen auf dem System, existierenden Nutzer.

%CurrentProfileDir% // since 4.11.2.1

kann statt *%UserProfileDir%* verwendet werden um *Files*-Sektionen zu erzeugen die sich genauso auch in *userLog-inScripten* verwenden lassen.

%UserProfileDir% or %CurrentProfileDir%

NT5: c:\Documents and Settings\%USERNAME%

NT6: c:\users\%USERNAME%

8.2.4 opsi-winst/opsi-script Pfad und Verzeichnisse [W/L]

%ScriptPath% or **%ScriptDir%** : Pfad des *opsi-winst/opsi-script* Skripts (ohne schließenden Backslash); mit Hilfe dieser Variable können die Dateien in Skripten relativ bezeichnet werden. Zum Testen können sie z.B. auch lokal gehalten werden.

%ScriptDrive% : Laufwerk, auf dem das ausgeführt *opsi-winst/opsi-script* Skript liegt (inklusive Doppelpunkt).

%WinstDir% : Pfad (ohne schließenden Backslash), in dem der aktive *opsi-winst/opsi-script* liegt.

%WinstVersion% : Versionsstring des laufenden *opsi-winst/opsi-script*.

%Logfile% : Der Name der Log-Datei, die der *opsi-winst/opsi-script* benutzt.

%opsiTmpDir% // since 4.11.4.3

Pfad zum Verzeichnis welches für Temporäre Dateien verwendet werden sollte. (Unter Windows: c:\opsi.org\tmp)

%opsiLogDir% // since 4.11.4.3

Pfad zum Verzeichnis welches für Log Dateien verwendet werden sollte. (Unter Windows: c:\opsi.org\log)

%opsiScriptHelperPath%

Entspricht: %ProgramFiles32Dir%\opsi.org\opsiScriptHelper

Pfad in dem Hilfsprogramme, Libraries und ähnliches zur Scriptausführung installiert sein können.

Seit 4.11.3.2

%opsidata% // since 4.12.0.12

Pfad zum Verzeichnis welches für opsi Daten Dateien verwendet werden sollte. (Unter Windows: c:\opsi.org\data)

%opsiapplog% // since 4.12.0.12

Pfad zum Verzeichnis welches für Log Dateien von Programmen welche im user context laufen verwendet wird. (Unter Windows: c:\opsi.org\applog)

Beispiel:

Der Code:

```
comment "Testing: "
message "Testing constants: "+"%"+"winstversion" +"%"
set $ConstTest$ = "%WinstVersion%"
set $InterestingFile$ = "%winstdir%\winst.exe"
if not (FileExists($InterestingFile$))
    set $InterestingFile$ = "%winstdir%\winst32.exe"
endif
set $INST_Resultlist$ = getFileInfoMap($InterestingFile$)
set $CompValue$ = getValue("file version with dots", $INST_Resultlist$ )
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

liefert folgenden Log:

```
comment: Testing:
message Testing constants: %winstversion%

Set $ConstTest$ = "4.10.8.3"
The value of the variable "$ConstTest$" is now: "4.10.8.3"

Set $InterestingFile$ = "N:\develop\delphi\winst32\trunk\winst.exe"
The value of the variable "$InterestingFile$" is now: "N:\develop\delphi\winst32\trunk\winst.
exe"
```

```

If
  Starting query if file exist ...
  FileExists($InterestingFile$) <<< result true
  not (FileExists($InterestingFile$)) <<< result false
Then
EndIf

Set $INST_Resultlist$ = getFileInfoMap($InterestingFile$)
  retrieving strings from getFileInfoMap [switch to loglevel 7 for debugging]

Set $CompValue$ = getValue("file version with dots", $INST_Resultlist$ )
  retrieving strings from $INST_Resultlist$ [switch to loglevel 7 for debugging]
  The value of the variable "$CompValue$" is now: "4.10.8.3"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed

Else
EndIf

```

8.2.5 Netzwerk Informationen [W/L]

%Host%: (Deprecated) Wert der Umgebungsvariable HOST. (Nicht mit **%HostId%** verwechseln)

%PCName%: Wert der Umgebungsvariable PCNAME oder wenn nicht vorhanden COMPUTERNAME. (Dies ist üblicherweise der Netbios Name)

%IPName%: Der DNS Name eines Computers. In vielen Fällen (aber nicht zwingend) ist dieser identisch mit dem netbios-Namen und damit auch identisch mit **%PCName%**. (Nur das der netbios-Namen üblicherweise in Großbuchstaben geschrieben wird.)

- **%IPAddress%**: (Deprecated) Liefert eine IP-Adresse eines Interface dieses Rechners. Verwenden Sie besser die Funktion **GetMyIpByTarget()**.
see also : [\[GetMyIpByTarget\]](#)

%Username%: Name des aktuellen Benutzers.

8.2.6 Service Daten [W/L]

%HostID%: FQDN des Clients.

Im opsi-Service-Kontext besser **%opsiserviceUser%** verwenden.

%opsiserviceURL%: Die opsi-Service URL des opsi config Servers (<https://<opsiserver>:4447>)

%opsiServer%: Name des opsi Config Servers abgeleitet von **%opsiserviceURL%**

%opsiDepotId%: Depot Server (FQDN) //since 4.11.4

%opsiserviceUser%: Die Benutzer ID für die es eine Verbindung zum opsi Service gibt. Im opsi-Service-Kontext der von opsi verwendete FQDN des Clients.

%opsiservicePassword%: Das für die Kommunikation mit dem Server verwendete Passwort. Das Passwort wird üblicherweise nicht geloggt.

`%installingProdName%`: Der Produktname (productId) für das der Service das laufende Skript aufruft. In dem Fall, dass das Skript nicht über den Service läuft, bleibt der String-Eintrag leer.

`%installingProdVersion%`: Ein String aus <Produktversion>-<Packageversion> für das der Service das laufende Skript aufruft. In dem Fall dass das Skript nicht über den Service läuft bleibt der String-Eintrag leer.

`%installingProduct%` : Product ID (abgekündigt).

8.3 String- (oder Text-) Variable [W/L]

8.3.1 Deklaration

String-Variable müssen vor ihrer Verwendung deklariert werden. Die Deklarationssyntax lautet

```
DefVar <variable name>
```

Beispielsweise

```
DefVar $MsVersion$
```

Erklärung:

- Die Variablennamen müssen nicht mit "\$" beginnen oder enden, diese Konvention erleichtert aber ihre Verwendung und vermeidet Probleme bei der Ersetzung der Variablen durch ihre Inhalte und ist daher **dringend** empfohlen.
- Die Deklaration von Variablen ist nur in den primären Sektionstypen (Actions-Sektion, sub-Sektionen sowie ProfileActions) möglich.
- Die Deklaration sollte nicht abhängig sein. Daher sollte die Deklaration auch nicht in Klammern in einer if – else - Konstruktion erfolgen. Da es sonst es passieren kann, dass ein DefVar-Anweisung nicht für eine Variable ausgeführt wird, aber die Variable in der if-Schleife ausgelesen wird und dann einen Syntax-Fehler produziert.
- Bei der Deklaration werden die Variablen mit dem leeren String ("") als Wert initialisiert.

Empfehlung:

- Alle Variablennamen sollten mit dem Zeichen \$ beginnen und enden.
- Alle Variablen sollten am Anfang des Skripts deklariert werden.

8.3.2 Wertzuweisung

In den primären Sektionstypen kann einer Variablen ein- oder mehrfach ein Wert zugewiesen werden. Die Syntax lautet:

```
Set <Variablenname> = <Value>
```

<Value> kann jeder String basierte Ausdruck sein (Beispiele dazu im Abschnitt Abschnitt 9.3).

```
Set $OS$ = GetOS
Set $NTVersion$ = "nicht bestimmt"

if $OS$ = "Windows_NT"
    Set $NTVersion$ = GetNTVersion
endif

DefVar $Home$
Set $Home$ = "n:\home\user name"
DefVar $MailLocation$
Set $MailLocation$ = $Home$ + "\mail"
```

8.3.3 Verwendung von Variablen in String-Ausdrücken

Eine Variable fungiert in den primären Sektionen als "Träger" eines Wertes. Zunächst wird sie deklariert und automatisch mit dem leeren String - also "" - initialisiert. Nach der Zuweisung eines Wertes mit dem Set-Befehl steht sie dann für diesen Wert.

In primären Sektionen, wie in der letzten Zeile des Beispiel-Codes zu sehen, kann die Variable selbst Teil von *opsi-winst/opsi-script* String-Ausdrücken werden.

```
Set $MailLocation$ = $Home$ + "\mail"
```

In der primären Sektion bezeichnet der Variablenname ein Objekt, das für einen String steht. Wenn die Variable hinzugefügt wird, steht diese für den ursprünglichen String.

In den sekundären Sektionen spielt dagegen ihr Name Platzhalter für die Zeichenfolge des von ihr repräsentierten Wertes:

8.3.4 Sekundäre und Primäre Sektion im Vergleich

Wenn eine sekundäre Sektion geladen wird, interpretiert der *opsi-winst/opsi-script* die Zeichenabfolge als Variablennamen und vergibt entsprechend die neuen Werten.

Beispiel:

Mit einer Kopieraktion in einer Files-Sektion soll eine Datei nach "n:\home\user name\mail\backup" kopiert werden.

Zuerst müsste das Verzeichnis \$MailLocation\$ gesetzt werden:

```
DefVar $Home$
DevVar $MailLocation$
Set $Home$ = "n:\home\user name"
Set $MailLocation$ = $Home$ + "\mail"
```

\$MailLocation\$ wäre dann
"n:\home\user name\mail"

In der primären Sektion würde man das Verzeichnis
"n:\home\user name\mail\backup"
durch die Variablen
\$MailLocation\$ + "\backup"
setzen.

Das gleiche Verzeichnis würde in der sekundären Sektion folgendermaßen aussehen:
"\$MailLocation\$\backup"

Ein grundsätzlicher Unterschied zwischen dem Variablenverständnis in der primären und sekundären Sektion ist, dass man in der primären Sektion einen verknüpften Ausdruck wie folgt formulieren kann:
\$MailLocation\$ = \$MailLocation\$ + "\backup"

Das bedeutet, dass \$MailLocation\$ zuerst einen initialen Wert und dann einen neuen Wert annimmt, in dem eine String zu dem initialen Wert addiert wird. Die Referenz der Variablen ist dynamisch und muss eine Entwicklung vollziehen.

In der sekundären Sektion ist ein solcher Ausdruck ohne Wert und würde eventuell einen Fehler verursachen, sobald \$MailLocation\$ durch die Verbindung mit einem festgelegten String ersetzt wird (bei allen virtuellen Vorgängen im selben Moment).

8.4 Variable für String-Listen [W/L]

Variable für String-Listen müssen vor ihrer anderweitigen Verwendung mit dem Befehl DefStringList deklariert werden, z.B.

DefStringList SMBMounts

String-Listen können z.B. die Ausgabe eines Shell-Programms einfangen und dann in vielfältiger Weise weiterverarbeitet und verwendet werden. Genauere Details dazu findet sich in dem Abschnitt Abschnitt [9.4](#) zur String-Listenverarbeitung.

Kapitel 9

Syntax und Bedeutung der primären Sektionen eines *opsi-winst/opsi-script* Skripts [W/L]

Wie bereits in Abschnitt 4 dargestellt, zeichnen sich die Actions-Sektion dadurch aus, dass sie den globalen Ablauf der Abarbeitung eines opsi-winst-Skripts beschreiben und insbesondere die Möglichkeit des Aufrufs von Unterprogrammen, sekundärer oder geschachtelter primärer Sektionen bieten.

Diese Unterprogramme heißen Sub-Sektionen – welche wiederum in der Lage sind, rekursiv weitere Sub-Sektionen aufzurufen.

Der vorliegende Abschnitt beschreibt den Aufbau und die Verwendungsweisen der primären Sektionen des *opsi-winst/opsi-script* Skripts.

9.1 Die primären Sektionen [W/L]

In einem Skript können vier Arten primärer Sektionen vorkommen:

- eine **Initial**-Sektion zu Beginn des Skripts (kann entfallen),
- danach eine **Actions**-Sektion sowie
- (beliebig viele) **Sub**-Sektionen.
- eine **ProfileActions** Sektion

Initial- und **Actions**-Sektion sind bis auf die Reihenfolge gleichwertig (Initial sollte an erster Stelle stehen). In der Initial Sektion sollten statische Einstellungen und -werte bestimmt werden (wie z.B. der Log-Level). Inzwischen empfehlen wir, die Initial Sektion aus Gründen der Übersichtlichkeit wegzulassen. In der Actions-Sektion ist die eigentliche Abfolge der vom Skript gesteuerten Programmaktionen beschrieben ist und kann als Hauptprogramm eines *opsi-winst/opsi-script* Skripts gesehen werden.

Sub-Sektionen sind syntaktisch mit der Initial- und der Actions-Sektion vergleichbar, werden aber über die **Actions**-Sektion aufgerufen. In ihnen können auch wieder weitere **Sub**-Sektionen aufgerufen werden.

Sub-Sektionen werden definiert, indem ein Name gebildet wird, der mit Sub beginnt, z.B. **Sub_InstallBrowser**. Der Name dient dann (in gleicher Weise wie bei den sekundären Sektionen) als Funktionsaufruf, der Inhalt der Funktion bestimmt sich durch eine Sektion betitelt mit dem Namen (im Beispiel eingeleitet durch **[Sub_InstallBrowser]**).

Sub-Sektionen zweiter oder höherer Anforderung (Sub von Sub usw.) können keine weiteren inneren Sektionen beinhalten, aber es können externe Unterprogramme aufgerufen werden (siehe dazu [Abschnitt "Aufrufe von Unterprogrammen"](#)).

**Achtung**

Wenn (geschachtelte) Sub-Sektionen in externe Dateien ausgelagert werden, müssen die aufgerufenen Sekundären Sektionen üblicherweise in der Datei untergebracht werden, aus der sie aufgerufen werden. Je nach verwendeter Komplexität des Syntax müssen sie evtl. **zusätzlich** auch in der Hauptdatei untergebracht werden.

`ProfileActions` Sektion kann in einem normalen Installations script als sub sektion mit speziellen Syntax Regeln dienen. Existiert diese Section in einem Script das als `userLoginScript` aufgerufen wurde, so ist diese Sektion der Programmstart (statt `Actions`). Siehe Kapitel *User Profile Management* im opsi-manual sowie Abschnitt 9.11

9.2 Parametrisierungsanweisungen für den *opsi-winst/opsi-script* [W/L]

Typisch für den Inhalt der Initial-Sektion sind diverse Parametrisierungen des opsi-winst. Das folgende Beispiel zeigt, wie darüber hinaus die Logging-Funktionen eingestellt werden können.

9.2.1 Beispiel

```
[Initial]
SetLogLevel=5
ExitOnError=false
ScriptErrorMessages=on
TraceMode=off
```

Dies bedeutet, dass

- der Detaillierungsgrad der Protokollierung auf Level 5 gesetzt wird,
- die Abarbeitung des Skripts bei Auftreten eines Fehlers in einer Sektion beendet wird,
- Skript-Syntaxfehler (in gesonderten Fenstern) angezeigt werden und schließlich
- der Einzelschrittmodus bei der Abarbeitung des Skripts deaktiviert bleibt.

Es handelt sich hier jeweils um den Defaultwert, der auch gültig ist, wenn die betreffende Anweisung fehlt.

Der Aufbau der Anweisungszeilen und ihre Bedeutung im Einzelnen:

9.2.2 Festlegung der Protokollierungstiefe

**Achtung**

Die alte Funktion `LogLevel=` ist ab Winst Version 4.10.3 abgekündigt. Um Rückwärtskompatibilität zu alten Skripten zu gewährleisten wird zu dem hiermit gesetzten Loglevel nochmal 4 hinzuaddiert.

Es gibt zwei ähnliche Varianten, um den Loglevel zu spezifizieren:

```
SetLogLevel = <Zahl>
```

```
SetLogLevel = <STRINGAUSDRUCK>
```

`SetLogLevel` definiert die Tiefe der Protokollierung der Operationen. Im ersten Fall kann die Nummer als Integer Wert oder als String-Ausdruck (vgl. Abschnitt 9.3) angegeben werden. Im zweiten Fall versucht der *opsi-winst/opsi-script* den String-Ausdruck als Nummer auszuwerten.

Es sind zehn Detaillierungsgrade wählbar von 0 bis 9

```

0 = nothing (absolut nichts)
1 = essential ("unverzichtbares")
2 = critical (unerwartet Fehler die zu einem Programmabbruch führen)
3 = error (Fehler, die kein laufendes Programm abbrechen)
4 = warning (das sollte man sich nochmal genauer anschauen)
5 = notice (wichtige Aussagen zu dem Programmverlauf)
6 = info (zusätzlich Informationen)
7 = debug (wichtige debug Meldungen)
8 = debug2 (mehr debug Informationen und Daten)
9 = confidential (Passwörter und andere sicherheitsrelevante Daten)

```

- Auf Loglevel 6 wird geloggt:
Alle Programmanweisungen, alle Wertzuweisungen zu Stringvariablen, die Ergebnisse kompletter boolescher Ausdrücke (hinter if)
- Auf Loglevel 7 wird geloggt:
Alle Zuweisungen zu Stringlisten Variablen, die Ausgaben von externen Prozessen, soweit diese nicht einer Stringliste zugewiesen werden, die Ergebnisse der Teilauswertung boolescher Ausdrücke (hinter if)
- Auf Loglevel 8 wird geloggt:
Stringlisten welche von Funktionen erzeugt werden, die Ausgaben von externen Prozessen, wenn diese einer Stringliste zugewiesen werden

Der Default ist "7".

see also : [\[opsi-script-configs_default_loglevel\]](#) see also : [\[opsi-script-configs_force_min_loglevel\]](#)

9.2.3 Benötigte *opsi-winst/opsi-script* Version [W/L]

Die Anweisung

```
requiredWinstVersion <RELATIONSSYMBOL> <ZAHLENSTRING>
```

z.B.

```
requiredWinstVersion >= "4.3"
```

lässt den *opsi-winst/opsi-script* überprüfen, ob die geforderte Versioneigenschaften vorliegt. Wenn nicht erscheint ein Fehlerfenster.

Dieses Feature gibt es erst ab *opsi-winst* Version 4.3 – bei früheren Versionen führt die noch unbekannt Anweisung einfach zu einer Syntaxfehlermeldung (vgl. auch den folgenden Abschnitt). Daher kann das Statement unabhängig von der aktuell benutzen *opsi-winst/opsi-script* Version benutzt werden so lange die erforderliche Version mindestens 4.3 ist.

Es gibt keinen Default.

9.2.4 Reaktion auf Fehler [W/L]

Zu unterscheiden sind zwei Sorten von Fehlern, die unterschiedlich behandelt werden müssen:

1. fehlerhafte Anweisungen, die der *opsi-winst/opsi-script* nicht "verstehen", d.h. deren Interpretation nicht möglich ist (syntaktischer Fehler),
2. aufgrund von "objektiven" Fehlersituationen scheiternde Anweisungen (Ausführungsfehler).

Normalerweise werden syntaktische Fehler in einem PopUp-Fenster für eine baldige Korrektur angezeigt, Ausführungsfehler werden in einer Log-Datei protokolliert und können später analysiert werden.

Das Verhalten des *opsi-winst/opsi-script* bei einem syntaktischen Fehler wird über die Konfiguration bestimmt.

- **ScriptErrorMessages** = <Wahrheitswert>
 Wenn der Wert *true* ist (Default), werden Syntaxfehler bzw. Warnhinweise zum Skripts als Message-Fenster auf dem Bildschirm angezeigt.
 Für <Wahrheitswert> kann außer *true* bzw. *false* hier zwecks einer intuitiveren Bezeichnung auch *on* bzw. *off* eingesetzt werden.
 Default=true
 siehe auch: [\[opsi-script-configs_ScriptErrorMessages\]](#)

- **FatalOnSyntaxError** = <Wahrheitswert>
 - *true* = (default) Bei einem Syntaxfehler wird das Script abgebrochen und *failed* zurückgeliefert. Dem Server wird die Meldung *Syntax Error* übergeben.
 - *false* = Bei einem Syntaxfehler wird das Script **nicht** abgebrochen.
 Der Syntaxfehler wird in jedem Fall als *Critical* in die Logdatei übernommen.
 In jedem Fall wird der Errorcounter um 1 erhöht.
 Seit 4.11.3.2
 In älteren Versionen wird weder gelogged noch abgebrochen.

- **FatalOnRuntimeError** = <Wahrheitswert>
 Ein *RuntimeError* ist ein Fehler in der Scriptlogik der zu einer verbotenen Operation führt. Ein Beispiel ist von einer Stringliste welche 2 Strings hat den 5. String zu fordern.
 - *true* = Bei einem *RuntimeError* wird das Script abgebrochen und *failed* zurückgeliefert. Dem Server wird die Meldung *Runtime Error* übergeben.
 - *false* = (default) Bei einem Syntaxfehler wird das Script **nicht** abgebrochen. Der *RuntimeError* wird als *Error* in die Logdatei übernommen und wird der Errorcounter um 1 erhöht.
 Seit 4.11.4.3

Die beiden folgenden Einstellungen steuern die Reaktion auf Fehler bei der Ausführung des Skripts.

- **ExitOnError** = <Wahrheitswert>
 Mit dieser Anweisung wird festgelegt, ob bei Auftreten eines Fehlers die Abarbeitung des Skripts beendet wird. Wenn <Wahrheitswert> *true* oder *yes* oder *on* gesetzt wird, terminiert das Programm, andernfalls werden die Fehler lediglich protokolliert (default).
- **TraceMode** = <Wahrheitswert>
 Wird *TraceMode* eingeschaltet (Default ist *false*), wird jeder Eintrag ins Protokoll zusätzlich in einem Message-Fenster auf dem Bildschirm angezeigt und muss mit einem OK-Schalter bestätigt werden.

9.2.5 Vordergrund [W]

- **StayOnTop** = <Wahrheitswert>

Mittels *StayOnTop* = *true* (oder = *on*) kann bestimmt werden, dass im Batchmodus das *opsi-winst/opsi-script* Fenster den Vordergrund des Bildschirms in Beschlag nimmt, sofern kein anderer Task den selben Status beansprucht. Im Dialogmodus hat der Wert der Variable keine Bedeutung.

**Achtung**

Nach Programmiersystem-Handbuch soll der Wert nicht im laufenden Betrieb geändert werden. Zur Zeit sieht es so aus, als wäre ein einmaliges (Neu-) Setzen des Wertes möglich, ein Rücksetzen auf den vorherigen Wert während des Programmlaufs dann aber nicht mehr.

StayOnTop steht per Default auf false damit verhindert wird das Fehlermeldungen eventuell nicht sichtbar sind, weil der *opsi-winst/opsi-script* im Vordergrund läuft.

9.2.6 Fenster Modus / Skin / Aktivitätsanzeige

- **SetSkinDirectory** <skindir> // [W/L]
Setzt das zu verwendende SkinDirectory und lädt den Skin Wird bei diesem Befehl ein leerer oder ungültiger Pfad angegeben, so wird der Defaultpfad verwendet.
Der Defaultpfad ist %WinstDir%\winstskin.

Beispiel:

```
SetSkinDirectory "%ScriptPath%\testskin"
sleepseconds 1
SetSkinDirectory ""
```

- **NormalizeWinst**
setzt das *opsi-winst/opsi-script* Fenster auf *normal* Modus.
- **IconizeWinst**
setzt das *opsi-winst/opsi-script* Fenster auf *minimierten* Modus.
- **MaximizeWinst**
setzt das *opsi-winst/opsi-script* Fenster auf *maximierten* Modus. // since 4.11.5.1
- **RestoreWinst**
setzt das *opsi-winst/opsi-script* Fenster auf deb letzten Modus.

AutoActivityDisplay = <boolean value> //since 4.11.4.7
(default=false);

Wenn true wird während des Laufs von externen Prozessen (winbatch,dosbatch,execwith Sektionen) ein (marquee) Fortschrittsbalken (der Endlos durch läuft) angezeigt.

siehe auch: [\[opsi-script-configs_AutoActivityDisplay\]](#)

9.3 String-Werte, String-Ausdrücke und String-Funktionen [W/L]

Ein String Ausdruck kann

- ein elementarer String-Wert
- ein verschachtelter String-Wert
- eine String-Variable
- eine Verknüpfung von String-Ausdrücken oder
- ein stringbasierter Funktionsaufruf sein.

9.3.1 Elementare String-Werte

Ein elementarer String-Wert ist jede Zeichenfolge, die von doppelten – " – oder von einfachen – ' – Anführungszeichen umrahmt ist:

```
"<Zeichenfolge>"
```

oder

```
'<Zeichenfolge>'
```

Zum Beispiel:

```
DefVar $BeispielString$
Set $BeispielString$ = "mein Text"
```

9.3.2 Strings in Strings („geschachtelte“ String-Werte)

Wenn in der Zeichenfolge Anführungszeichen vorkommen, muss zur Umrahmung die andere Variante des Anführungszeichens verwendet werden.

```
DefVar $Zitat$
Set $Zitat$ = 'er sagte "Ja"'
```

Zeichenfolgen, innerhalb derer möglicherweise bereits Schachtelungen von Anführungszeichen vorkommen, können mit EscapeString: <Abfolge von Buchstaben> gesetzt werden. Z.B. bedeutet

```
DefVar $MetaZitat$
Set $MetaZitat$ = EscapeString: Set $Zitat$ = 'er sagte "Ja"'
```

dass auf der Variablen \$MetaZitat\$ am Ende exakt die Folge der Zeichen nach dem Doppelpunkt von EscapeString (inklusive des führenden Leerzeichens) steht, also

```
Set $Zitat$ = 'er sagte "Ja"'
```

9.3.3 String-Verknüpfung

String-Verknüpfung werden mit dem Pluszeichen ("+") gemacht

```
<String expression> + <String expression>
```

Beispiel:

```
DefVar $String1$
DefVar $String2$
DefVar $String3$
DefVar $String4$
Set $String1$ = "Mein Text"
Set $String2$ = "und"
Set $String3$ = "dein Text"
Set $String4$ = $String1$ + " " + $String2$ + " " + $String3$
```

\$String4\$ hat dann den Wert "Mein Text und dein Text".

9.3.4 String-Ausdrücke

Eine String-Variable der primären Sektion "beinhaltet" einen String-Wert. Ein String-Ausdruck kann einen elementaren String vertreten. Wie ein String gesetzt und definiert wird findet sich in Abschnitt Abschnitt 8.3.

Die folgenden Abschnitte zeigen die Variationsmöglichkeiten von String-Funktionen.

9.3.5 String-Funktionen zur Ermittlung des Betriebssystemtyps

- **GetOS** [W/L]
Die Funktion ermittelt das laufende Betriebssystem. Derzeit liefert sie einen der folgenden Werte: "Windows_16"
"Windows_95" (auch bei Windows 98 und ME) "Windows_NT" (bei Windows NT - Windows 8) "Linux"
- **GetNtVersion** [W]
(abgekündigt: use GetMSVersionInfo)
Für ein Betriebssystem mit Windows_NT ist eine Type-Nummer und eine Sub Type-Nummer charakteristisch. GetNtVersion gibt den genauen Sub Type-Namen aus. Mögliche Werte sind
"NT3"
"NT4"
"Win2k" (Windows 5.0)
"WinXP" (Windows 5.1)
"Windows Vista" (Windows 6)

Bei höheren (als 6.*) Windows-Versionen werden die Versionsnummern (5.2, ... bzw. 6.0 ..) ausgegeben. Z.B. für Windows Server 2003 R2 Enterprise Edition erhält man

"Win NT 5.2" (Windows Server 2003)

In dem Fall, dass kein NT-Betriebssystem vorliegt, liefert die Funktion als Fehler-Wert:

"Kein OS vom Typ Windows NT"

- **GetMsVersionInfo** [W]
gibt für Systeme des Windows NT Typs die Information über die Microsoft Version als API aus, z.B. produziert ein Windows XP System das Ergebnis
"5.1"

Tabelle 9.1: Windows Versionen

GetMsVersionInfo	Windows Version
5.0	Windows 2000
5.1	Windows XP (Home, Prof)
5.2	XP 64 Bit, 2003, Home Server, 2003 R2
6.0	Vista, 2008
6.1	Windows 7, 2008 R2
6.2	Windows 8, 2012
6.3	Windows 8.1, 2012 R2
10.0	Windows 10

siehe auch **GetMsVersionMap**

- **GetSystemType** [W/L]
prüft die Architektur des installierten Betriebssystems. Im Fall eines 64 Bit-Betriebssystems ist der ausgegebene Wert *64 Bit System* oder *x86 System*.
- **getLinuxDistroType** [L]
liefert den Typ der laufenden Linuxdistribution. Mögliche Werte:
 - *debian* (Debian / Ubuntu → use apt-get)
 - *redhat* (RedHat / CentOS → use yum)
 - *suse* (→ use zypper) (siehe auch **getLinuxVersionMap**)

9.3.6 String-Funktionen zur Ermittlung von Umgebungs- und Aufrufparametern [W/L]

- `EnvVar (<string>)` [W/L]
Die Funktion liefert den aktuellen Wert einer Umgebungsvariablen. Z.B. wird durch `EnvVar ("Username")` der Name des eingeloggten Users ermittelt.
- `ParamStr` [W/L]
Die Funktion gibt den String aus, der im Aufruf von *opsi-winst/opsi-script* nach dem optionalen Kommandozeilenparameter `/parameter` folgt. Ist der Kommandozeilenparameter nicht verwendet, liefert `ParamStr` den leeren String.
- `getLastExitCode` [W/L]
Die String-Funktion `getLastExitCode` gibt den ExitCode des letzten Prozessaufrufs der vorausgehenden *WinBatch / DosBatch / ExecWith* Sektion aus.
Der Aufruf anderer opsi-winst Befehle (wie z.B. einer *Files* Sektion) verändert den gefundenen ExitCode nicht.
Bei *DosBatch* und *ExecWith* Sektionen erhalten wir den Exitcode des Interpreters. Daher muss in der Regel der gewünschte Exitcode in der Sektion explizit übergeben werden.

Beispiel:

```
DosInAnIcon_exit1
set $ConstTest$ = "1"
set $CompValue$ = getLastExitCode
if ($ConstTest$ = $CompValue$)
    comment "DosBatch / DosInAnIcon  exitcode passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "DosBatch / DosInAnIcon  exitcode failed"
endif

[DosInAnIcon_exit1]
rem create an errolevel= 1
VERIFY OTHER 2> NUL
echo %ERRORLEVEL%
exit %ERRORLEVEL%
```

- `GetUserSID(<Windows Username>)` [W]
gibt die SID für den übergebenen Benutzer an (möglich mit der Domäne als Vorspann in Form von DOMAIN\USER).
- `GetUsercontext` [W]
Die Funktion gibt den String aus, der im Aufruf von *opsi-winst/opsi-script* nach dem optionalen Kommandozeilenparameter `/usercontext` folgt. Ist der Kommandozeilenparameter nicht verwendet, liefert `GetUsercontext` den leeren String.

9.3.7 Werte aus der Windows-Registry lesen und für sie aufbereiten [W]

- `getRegistryValue (<keystr>, <varstr> [, <access str>])` : string //since 4.12.0.16 [W]
versucht in der Registry den als `<keystr>` übergebenen String-Wert als Registrykey zu öffnen und dort den Wert `<varstr>` zu lesen und als String zurückzugeben.
Wenn `<keystr>` bzw. die Variable `<varstr>` nicht existieren, wird eine Warnung in das Logfile geschrieben und der Leerstring als Defaultwert zurückgegeben.

Wenn der übergebene <varstr> leer ist, so wird der Standardwert des Keys ausgelesen.

Die Zugriffsart ist per Default `sysnative`. Über den optionalen dritten Parameter kann die Zugriffsart auch explizit angegeben werden. Dabei muss der übergebene <access str> einer der folgenden Werte sein: `32bit`, `sysnative`, `64bit`.

(siehe auch: [Kapitel 64 Bit-Unterstützung](#))

Beispiele:

```
getRegistryValue("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon", "
Shell")

getRegistryValue("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon", "
Shell", "64bit")
```

- `GetRegistryStringValue (<string>)`
versucht den übergebenen String-Wert als einen Ausdruck der Form
`[KEY] X`
zu interpretieren; im Erfolgsfall liest die Funktion den (String-) Wert zum Variablennamen `X` des Schlüssels `KEY` der Registry aus. Wird `X` nicht übergeben so wird der Standardwert des Keys ausgelesen.
Wenn `KEY` bzw. die Variable `X` nicht existieren, wird eine Warnung in das Logfile geschrieben und der Leerstring als Defaultwert zurückgegeben.
siehe auch : [\[getRegistryValue\]](#)

Zum Beispiel ist

```
GetRegistryStringValue (" [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
Winlogon] Shell")
```

üblicherweise "Explorer.exe", das default Windows Shell Programm (aber es könnten auch andere Programme als Shell eingetragen sein.)

Zum Beispiel liefert

```
Set $CompValue$ = GetRegistryStringValue32 (" [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-
test\test-4.0] ")
```

wenn der Key vorher mit dem Standardeintrag *standard entry* erzeugt wurde, den folgenden Logeintrag:

```
Registry started with redirection (32 Bit)
Registry key [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-test\test-4.0] opened
Key closed
The value of the variable "$CompValue$" is now: "standard entry"
```

- `GetRegistryStringValue32(<string>)` → siehe [Kapitel 64 Bit-Unterstützung](#)
siehe auch : [\[getRegistryValue\]](#)
- `GetRegistryStringValue64(<string>)` → siehe [Kapitel 64 Bit-Unterstützung](#)
siehe auch : [\[getRegistryValue\]](#)
- `GetRegistryStringValueSysNative(<string>)` → siehe [Kapitel 64 Bit-Unterstützung](#)
siehe auch : [\[getRegistryValue\]](#)

- `RegString(<string>)`
wird vor allem benötigt, um Dateinamen in die Form zu wandeln, in der sie in die Registry eingetragen werden, das heißt, jeder Backslash des Arguments der Funktion wird verdoppelt. Z.B. liefert

```
RegString ("c:\windows\system\")
```

den Wert

```
"c:|\windows|system|"
```

9.3.8 Werte aus Ini-Dateien lesen [W/L]

Es gibt – aus historischen Gründen – zwei Funktionen, um Werte aus Ini-Dateien zu lesen.

Als Ini-Datei wird dabei jede in "Sektionen" gegliederte Textdatei der Form

```
[Sektion1]
Varname1=Wert1
Varname2=Wert2
...
[Sektion2]
...
```

bezeichnet.

Die allgemeinste Funktion liest den Key-Wert aus einer Sektion der ini-Datei aus. Jeder Parameter kann als ein willkürlicher String-Ausdruck ausgegeben werden:

- `GetValueFromInifile (<FILE>, <SECTION>, <KEY>, <DEFAULTVALUE>)` [W/L]
Die Funktion öffnet die Ini-Datei namens FILE und sucht in deren Sektion SECTION den KEY (auch Variable genannt). Wenn diese Operation erfolgreich sind, wird der zum KEY gehörende Wert zurückgegeben, andernfalls DEFAULTVALUE.
- `GetIni (<Stringausdruck> [<character sequence>] <character sequence>)`
(abgekündigt: use `GetValueFromInifile`)
Diese Funktion unterstützt eine Schreibweise, die sehr eng an die Schreibweise der Ini-Datei selbst angelehnt ist. Dabei kann allerdings nur der Dateiname durch einen String-Ausdruck dargestellt werden, die anderen Größen müssen explizit angegeben sein:
Der <Stringausdruck> wird als Dateiname ausgelesen, der Erste <character sequence> als Sektionsname, der Zweite als Schlüsselname.
GetIni("MEINEINIDATEI"[meinesektion] meinkey)
gibt diese selben Werte zurück wie
GetValueFromInifile("MEINEINIDATEI", "meinesektion", "meinkey", "")

9.3.9 Produkt Properties auslesen [W/L]

- `GetProductProperty (<PropertyName>, <DefaultValue>)` [W/L]
wobei <PropertyName> und <DefaultValue> String-Ausdrücke sind. Die Funktion liefert die client-spezifischen Property-Werte für das aktuell installierte Produkt aus.
Auf diese Weise können PC-spezifische Varianten einer Installation konfiguriert werden.
Außerhalb des opsi-service Kontextes oder wenn aus anderen Gründen der Aufruf fehlschlägt, wird der angegebene Defaultwert zurückgegeben.

So wurde beispielsweise die opsi UltraVNC Netzwerk Viewer Installation mit folgenden Produkt Properties konfiguriert:

- `viewer = <yes> | <no>`

- `policy = <factory_default> |`

Innerhalb des Installationsskript werden die ausgewählten Werte wie folgt abgerufen

```
GetProductProperty("viewer", "yes")
GetProductProperty("policy", "factory_default")
```

- `GetConfidentialProductProperty (<PropertyName>, <DefaultValue>)` //since 4.11.5.2
verhält sich wie `GetProductProperty` nur das der resultierende Wert als confidential string behandelt und damit nicht gelogged wird.
Sinnvoll um z.B. Passwörter aus einem Produktproperty abzufragen. Siehe auch `SetConfidential`
- `IniVar(<PropertyName>)`
(abgekündigt: use `GetProductProperty`)

9.3.10 Informationen aus etc/hosts entnehmen [W/L]

- `GetHostsName(<string>)` [W/L]
liefert den Host-Namen zu einer gegebenen IP-Adresse entsprechend den Angaben in der Host-Datei (die, falls das Betriebssystem laut Environment-Variablen OS "Windows_NT" ist, im Verzeichnis "%system-root%\system32\drivers\etc\" gesucht wird, andernfalls in "C:\Windows\").
- `GetHostsAddr(<string>)` [W/L]
liefert die IP-Adresse zu einem gegebenen Host- bzw. Aliasnamen entsprechend der Auflösung in der Host-Datei.

9.3.11 String-Verarbeitung [W/L]

- `ExtractFilePath(<string>)` [W/L]
interpretiert den übergebenen String als Datei- bzw. Pfadnamen und gibt den Pfadanteil (den String-Wert bis einschließlich des letzten "\" zurück).
- `StringSplit (`STRINGWERT1, STRINGWERT2, INDEX)``
(abgekündigt: use `splitString / takestring`)
- `takeString(<index>, <list>)` [W/L]
liefert aus der String-Liste `<list>` den String mit dem Index `<index>`.
Häufig verwendet in Kombination mit `splitString`: `takeString(<index>, splitString(<string1>, <string2>))`
(siehe den Abschnitt [String-Listenverarbeitung](#)).
Das Ergebnis ist, dass `<string1>` in Stücke zerlegt wird, die jeweils durch `<string2>` begrenzt sind, und das Stück mit `<index>` (Zählung bei 0 beginnend) genommen wird.

Zum Beispiel ergibt

```
takeString(3, splitString ("\\server\share\directory", "\"))
```

den Wert `"share"`,

denn mit `\` zerlegt sich der vorgegebene String Wert in die Folge:

Index 0 - "" (Leerstring), weil vor dem ersten "\" nichts steht

Index 1 - "" (Leerstring), weil zwischen erstem und zweitem "\" nichts steht

Index 2 - "server"

Index 3 - "share"

Index 4 - "directory"

`takestring` zählt abwärts, wenn der Index negativ ist, beginnend mit der Zahl der Elemente.

Deswegen

```
takestring(-1, $list1$)
```

bedeutet das letzte Element der String-Liste \$list1\$.

- **SubstringBefore**(<string1>, <string2>)
(abgekündigt: use splitString / takestring) liefert das Anfangsstück von <string1>, wenn <string2> das Endstück ist. Z.B. hat

```
SubstringBefore ("C:\programme\staroffice\program\soffice.exe", "\program\soffice.exe")
```

den Wert "C:\programme\staroffice".

- **getIndexFromListByContaining**(<list> : stringlist, <search string> : string)` : <number> : string //since 4.12.0.13 [W/L]
Liefert den Index (als String) von dem ersten String in <list> in dem <search string> vorkommt. Wird <search string> nicht gefunden so wird ein Leerstring geliefert.
Die Überprüfung auf Gleichheit ist nicht Case Sensitive.
- **takeFirstStringContaining**(<list>, <search string>) [W/L]
Liefert den ersten String einer Liste der <search string> enthält. Liefert einen leeren String, wenn kein passender String gefunden wird.
- **Trim**(<string>) [W/L]
Schneidet Leerzeichen am Anfang und Ende des <string> ab.
- **lower**(<string>) [W/L]
liefert <string> in Kleinbuchstaben
- **upper**(<string>) [W/L]
liefert <string> in Großbuchstaben
- **contains**(<str>, <substr>) [W/L]
Boolsche Funktion welche *true* liefert wenn <substr> in <str> enthalten ist. Die Funktion arbeitet case sensitive.
Seit 4.11.3
Beispiel:

```
set $ConstTest$ = "1xy451Xy451XY45"
set $CompValue$ ="xy"
if contains($ConstTest$, $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $CompValue$ ="xY"
if not(contains($ConstTest$, $CompValue$))
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

- `stringReplace(<string>, <oldPattern>, <newPattern>)` [W/L]
Liefert einen String in dem in <string>, <oldPattern> durch <newPattern> ersetzt ist. Die Funktion arbeitet nicht case sensitive.
Seit 4.11.3
Beispiel:

```
set $ConstTest$ = "123451234512345"
set $CompValue$ = stringReplace("1xy451Xy451XY45", "xy", "23")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

- `strLength(<string>)` [W/L]
Liefert Anzahl der Zeichen in <string>
Seit 4.11.3
Beispiel:

```
set $tmp$ = "123456789"
set $ConstTest$ = "9"
set $CompValue$ = strLength($tmp$)
if $ConstTest$ = $CompValue$
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

- `strPos(<string>, <sub string>)` [W/L]
Liefert die Position des ersten Vorkommens von <sub string> in <string>. Wird <sub string> nicht gefunden so liefert die Funktion "0". Die Funktion arbeitet case sensitive.
Seit 4.11.3
Beispiel:

```
set $tmp$ = "1xY451Xy451xy45"
set $ConstTest$ = "7"
set $CompValue$ = strPos($tmp$, "Xy")
if $ConstTest$ = $CompValue$
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $tmp$ = lower("1xY451Xy451xy45")
set $ConstTest$ = "2"
set $CompValue$ = strPos($tmp$, lower("xy"))
if $ConstTest$ = $CompValue$
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

- `strPart(<string>, <start pos>, <number of chars>)` [W/L]
Liefert einen Teilstring von <string> beginnend mit <start pos> und <number of chars> lang. Wenn ab <str pos> weniger als <number of chars> vorhanden sind, so wird der String bis zum Ende geliefert.
Die Zählung der Zeichen beginnt mit 1.
Seit 4.11.3
Beispiel:

```

set $tmp$ = "123456789"
set $ConstTest$ = "34"
set $CompValue$ = strPart($tmp$, "3", "2")
if $ConstTest$ = $CompValue$
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $tmp$ = "123456789"
set $ConstTest$ = "56789"
set $CompValue$ = strPart($tmp$, strPos($tmp$, "56"), strLength($tmp$))
if $ConstTest$ = $CompValue$
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

```

- `unquote(<string>, <quote-string>)` [W/L]
Wenn <string> mit dem Anführungszeichen <quote-string> versehen ist so liefert diese Funktion <string> ohne Anführungszeichen
Von <quote-string> wird nur das erste Zeichen berücksichtigt, wobei führende Whitespaces ignoriert werden.
Seit 4.11.2.1

```

set $ConstTest$ = "b"
    set $CompValue$ = unquote('`b`', '`')
    comment "compare values"
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif
    comment "double quote"
    set $ConstTest$ = "b"
    set $CompValue$ = unquote('"b"', '"')
    comment "compare values"
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif
    comment "quote string will be trimmed and then only the first char is used"
    comment "note: brackets are different chars"
    set $ConstTest$ = "b]"
    set $CompValue$ = unquote("[b]", "[{ ")

```

```

comment "compare values"
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
comment "not usable to remove brackets"
set $ConstTest$ = "b]"
set $CompValue$ = unquote("[b]", "[")
set $CompValue$ = unquote($CompValue$, "]"")
set $CompValue$ = unquote("[b]", "]"")
set $CompValue$ = unquote($CompValue$, "["")
set $CompValue$ = unquote(unquote("[b]", "[")", "]"")
comment "compare values"
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
comment "if string not quoted it will be come back without changes"
set $ConstTest$ = "b"
set $CompValue$ = unquote("b", "'")
comment "compare values"
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

```

`unquote2(<string>, <quote-string>)` // since 4.11.5.2 [W/L]

Funktioniert wie `unquote(<string>, <quote-string>)` mit folgenden Unterschieden:

Wenn `<quote-string>` ein Zeichen lang ist, wird diese Zeichen als erwartetes *Quote Zeichen* für den Anfang und das Ende von `<string>` genommen. Wenn `<quote-string>` zwei Zeichen lang ist, wird das erste Zeichen als erwartetes *Quote Zeichen* für den Anfang und das zweite Zeichen als erwartetes *Quote Zeichen* für den Ende von `<string>` genommen. Beispiel: `"()`"

Die Funktion liefert `<string>` unverändert zurück, wenn nicht sowohl für Anfang und Ende die erwarteten *Quote Zeichen* gefunden werden.

- `HexStrToDecStr(<string>)` [W/L]
wandelt einen String mit einem hexadezimalen Zahlwert in einen String mit dem entsprechenden decimalen Wert um. Enthält der Eingabe String führende Zeichen wie `0x` oder `$` werden diese ignoriert.
Im Fehlerfall: Leerstring
- `DecStrToHexStr (<decstring>, <hexlength>)` [W/L]
liefert einen `<hexlength>` langen String mit der hexadezimalen Darstellung von `<decstring>` zurück, wenn dieser die dezimale Darstellung eines Intergerwertes war. Im Fehlerfall: Leerstring
- `base64EncodeStr(<string>)` [W/L]
liefert einen String mit dem base64 encodedten Wert von `<string>`.

- `base64DecodeStr(<string>)` [W/L]
liefert einen String mit dem base64 decodedten Wert von <string>.

```
message "base64EncodeStr"
set $ConstTest$ = "YWJjZGVm"
set $tmp$ = "abcdef"
set $CompValue$ = base64EncodeStr($tmp$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

comment ""
comment "-----"
comment "Testing: "
message "base64DecodeStr"
set $ConstTest$ = "abcdef"
set $tmp$ = "YWJjZGVm"
set $CompValue$ = base64DecodeStr($tmp$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

- `encryptStringBlow(<keystring>,<datastring>)` : `string` //since 4.11.6 [W/L]
Verschlüsselt <datastring> mit dem Key <keystring> unter Verwendung von Blowfish und liefert den verschlüsselten Wert zurück.
- `decryptStringBlow(<keystring>,<datastring>)` : `string` //since 4.11.6 [W/L]
Entschlüsselt <datastring> mit dem Key <keystring> unter Verwendung von Blowfish und liefert den entschlüsselten Wert zurück.

```
set $ConstTest$ = "This string is very secret"
set $ConstTest$ = encryptStringBlow("linux123",$ConstTest$)
set $ConstTest$ = decryptStringBlow("linux123",$ConstTest$)
set $CompValue$ = "This string is very secret"
if ($ConstTest$ = $CompValue$)
    comment "cryptStringBlow passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing cryptStringBlow failed"
endif
```

- `md5sumFromFile(<path to file>)` : `string` //since 4.11.6 [W/L]
Liefert die md5summe der unter <path to file> gefundenen Datei zurück.
Im Fehlerfall ist der Rückgabewert ein Leerstring.

```

set $ConstTest$ = md5sumFromFile("%ScriptPath%\test-files\crypt\dummy.msi")
set $CompValue$ = strLoadTextFile("%ScriptPath%\test-files\crypt\dummy.msi.md5")
if ($ConstTest$ = $CompValue$)
    comment "md5sumFromFile passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing md5sumFromFile failed"
endif

```

- `reencodestr(<str>, <from>, <to>)` //since 4.11.4.2 [W/L]
liefert den String <str> mit dem encoding <to> zurück wobei davon ausgegangen wird das <str> gemäß <from> encoded war. <from> und <to> sind dabei encodings wie sie im Kapitel Abschnitt 6.3 aufgelistet sind.
- `strLoadTextFile (<filename>)` //since 4.11.4.6 [W/L]
liefert die erste Zeile von <filename> als String.
- `strLoadTextFileWithEncoding (<filename> , <encoding>) : string` [W/L]
liefert die erste Zeile von <filename> als String reencoded von <encoding>.
- `GetShortWinPathName(<longpath string>)` //since 4.11.5.2 [W]
Liefert den Shortpath (8.3) von <longpath string>. Lässt sich für <longpath string> kein Shortpath finden, so liefert die Funktion einen leeren String. + Beispiel: `GetShortWinPathName("C:\Program Files (x86)")` liefert "C:\PROGRA-2"

9.3.12 Weitere String-Funktionen

- `RandomStr` [W/L]
liefert Zufallsstrings (der Länge 10), die aus Klein- und Großbuchstaben sowie Ziffern bestehen. Genauer: 2 Kleinbuchstaben, 2 Großbuchstaben, 2 Sonderzeichen und 4 Ziffern. Die möglichen Sonderzeichen sind:
`!,$, (,), *, +, /, :, ;, =, ?, [,], {, }, ^, ~, $, °`
- `CompareDotSeparatedNumbers(<string1>, <string2>)` [W/L]
vergleicht zwei Strings vom Typ <zahl>[.<zahl>[.<zahl>[.<zahl>]]] und liefert "0" bei Gleichheit, "1" wenn string1 größer ist und "-1" wenn string1 kleiner ist.
siehe auch boolsche funktion `CompareDotSeparatedNumbers`

Beispiel:

Der Code:

```

comment "Testing: "
message "CompareDotSeparatedNumbers"
set $string1$ = "1.2.3.4.5"
set $string2$ = "1.2.3.4.5"
set $ConstTest$ = "0"
set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
    comment "$string1$+" is equal to "+$string2$
else
    set $TestResult$ = "not o.k."

```

```

        LogWarning "failed"
    endif

    set $string1$ = "1.2.31.4.5"
    set $string2$ = "1.2.13.4.5"
    set $ConstTest$ = "1"
    set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
    if ($ConstTest$ = $CompValue$)
        comment "passed"
        comment $string1$+ " is higher then "+$string2$
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif

    set $string1$ = "1.2.3.4.5"
    set $string2$ = "1.2.13.4.5"
    set $ConstTest$ = "-1"
    set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
    if ($ConstTest$ = $CompValue$)
        comment "passed"
        comment $string1$+ " is lower then "+$string2$
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif

    comment ""
    comment "-----"
    comment "Testing: "
    message "CompareDotSeparatedStrings"
    set $string1$ = "1.a.b.c.3"
    set $string2$ = "1.a.b.c.3"
    set $ConstTest$ = "0"
    set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
    if ($ConstTest$ = $CompValue$)
        comment "passed"
        comment $string1$+ " is equal to "+$string2$
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif
endif

```

liefert folgenden Log:

```

comment: Testing:
message CompareDotSeparatedNumbers

Set $string1$ = "1.2.3.4.5"
The value of the variable "$string1$" is now: "1.2.3.4.5"

Set $string2$ = "1.2.3.4.5"
The value of the variable "$string2$" is now: "1.2.3.4.5"

Set $ConstTest$ = "0"
The value of the variable "$ConstTest$" is now: "0"

```

```
Set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
The value of the variable "$CompValue$" is now: "0"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
  comment: 1.2.3.4.5 is equal to 1.2.3.4.5

Else
EndIf

Set $string1$ = "1.2.31.4.5"
The value of the variable "$string1$" is now: "1.2.31.4.5"

Set $string2$ = "1.2.13.4.5"
The value of the variable "$string2$" is now: "1.2.13.4.5"

Set $ConstTest$ = "1"
The value of the variable "$ConstTest$" is now: "1"

Set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
The value of the variable "$CompValue$" is now: "1"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
  comment: 1.2.31.4.5 is higher then 1.2.13.4.5

Else
EndIf

Set $string1$ = "1.2.3.4.5"
The value of the variable "$string1$" is now: "1.2.3.4.5"

Set $string2$ = "1.2.13.4.5"
The value of the variable "$string2$" is now: "1.2.13.4.5"

Set $ConstTest$ = "-1"
The value of the variable "$ConstTest$" is now: "-1"

Set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
The value of the variable "$CompValue$" is now: "-1"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
  comment: 1.2.3.4.5 is lower then 1.2.13.4.5

Else
EndIf
```

- `CompareDotSeparatedStrings (<string1>, <string2>)` [W/L]
vergleicht zwei Strings vom Typ `<string>.<string>[.<string>[.<string>]]` und liefert "0" bei Gleichheit, "1" wenn `string1` größer ist und "-1" wenn `string1` kleiner ist. Der Vergleich ist nicht Case sensitive.
siehe auch boolsche funktion `CompareDotSeparatedStrings`

Beispiel:

Der Code:

```

comment "Testing: "
message "CompareDotSeparatedStrings"
set $string1$ = "1.a.b.c.3"
set $string2$ = "1.a.b.c.3"
set $ConstTest$ = "0"
set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
    comment $string1$+" is equal to "+$string2$
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "1.a.b.c.3"
set $string2$ = "1.A.B.C.3"
set $ConstTest$ = "0"
set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
    comment $string1$+" is equal to "+$string2$
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "1.a.cb.c.3"
set $string2$ = "1.a.b.c.3"
set $ConstTest$ = "1"
set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
    comment $string1$+" is higher then "+$string2$
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "1.a.ab.c.3"
set $string2$ = "1.a.b.c.3"
set $ConstTest$ = "-1"
set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
    comment $string1$+" is lower then "+$string2$
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

```

```

set $string1$ = "1.2.13.4.5"
set $string2$ = "1.2.3.4.5"
set $ConstTest$ = "-1"
set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
    comment $string1$+ " is lower then "+$string2$
    comment "using CompareDotSeparatedStrings give wrong results on numbers"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "1.2.3.4.5"
set $string2$ = "1.2.13.4.5"
set $ConstTest$ = "1"
set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
    comment $string1$+ " is higher then "+$string2$
    comment "using CompareDotSeparatedStrings give wrong results on numbers"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

```

liefert folgenden Log:

```

comment: Testing:
message CompareDotSeparatedStrings

Set $string1$ = "1.a.b.c.3"
  The value of the variable "$string1$" is now: "1.a.b.c.3"

Set $string2$ = "1.a.b.c.3"
  The value of the variable "$string2$" is now: "1.a.b.c.3"

Set $ConstTest$ = "0"
  The value of the variable "$ConstTest$" is now: "0"

Set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "0"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
  comment: 1.a.b.c.3 is equal to 1.a.b.c.3

Else
EndIf

Set $string1$ = "1.a.b.c.3"
  The value of the variable "$string1$" is now: "1.a.b.c.3"

```

```
Set $string2$ = "1.A.B.C.3"
  The value of the variable "$string2$" is now: "1.A.B.C.3"

Set $ConstTest$ = "0"
  The value of the variable "$ConstTest$" is now: "0"

Set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "0"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
  comment: 1.a.b.c.3 is equal to 1.A.B.C.3

Else
EndIf

Set $string1$ = "1.a.cb.c.3"
  The value of the variable "$string1$" is now: "1.a.cb.c.3"

Set $string2$ = "1.a.b.c.3"
  The value of the variable "$string2$" is now: "1.a.b.c.3"

Set $ConstTest$ = "1"
  The value of the variable "$ConstTest$" is now: "1"

Set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "1"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
  comment: 1.a.cb.c.3 is higher then 1.a.b.c.3

Else
EndIf

Set $string1$ = "1.a.ab.c.3"
  The value of the variable "$string1$" is now: "1.a.ab.c.3"

Set $string2$ = "1.a.b.c.3"
  The value of the variable "$string2$" is now: "1.a.b.c.3"

Set $ConstTest$ = "-1"
  The value of the variable "$ConstTest$" is now: "-1"

Set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "-1"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
```

```

comment: passed
comment: 1.a.ab.c.3 is lower then 1.a.b.c.3

Else
EndIf

Set $string1$ = "1.2.13.4.5"
The value of the variable "$string1$" is now: "1.2.13.4.5"

Set $string2$ = "1.2.3.4.5"
The value of the variable "$string2$" is now: "1.2.3.4.5"

Set $ConstTest$ = "-1"
The value of the variable "$ConstTest$" is now: "-1"

Set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
The value of the variable "$CompValue$" is now: "-1"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
  comment: 1.2.13.4.5 is lower then 1.2.3.4.5
  comment: using CompareDotSeparatedStrings give wrong results on numbers

Else
EndIf

Set $string1$ = "1.2.3.4.5"
The value of the variable "$string1$" is now: "1.2.3.4.5"

Set $string2$ = "1.2.13.4.5"
The value of the variable "$string2$" is now: "1.2.13.4.5"

Set $ConstTest$ = "1"
The value of the variable "$ConstTest$" is now: "1"

Set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
The value of the variable "$CompValue$" is now: "1"

If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
  comment: 1.2.3.4.5 is higher then 1.2.13.4.5
  comment: using CompareDotSeparatedStrings give wrong results on numbers

Else
EndIf

```

- `getDiffTimeSec` [W/L]
liefert einen String mit dem ganzzahligen Wert der vergangenen Sekunden seit dem letzten Aufruf von `marktime`.
Seit Version 4.11.3.1

- `timeStampAsFloatStr` : `string` (Floating Number - format: *days.decimal days*) //since 4.11.6 [W/L]
Liefert einen aktuellen Timestamp als Fließkommazahlstring. Dabei ist Zahl vor dem Komma die Tage seit dem 30. Dezember 1899. Nach dem Komma kommt die Zeit in Bruchteilen des Tages.
Just for Fun: Warum nicht der 31. Dezember 1899: ?
siehe <http://www.delphibasics.co.uk/RTL.asp?Name=TDateTime>
- `SidToName(<well known sid>)` [W]
liefert einen String mit dem lokalisierten Namen der mit <well known sid> bezeichneten Gruppe. Zum Beispiel für *S-1-5-32-544* je nach Lokalisierung des Betriebssystems *Administratoren* oder *Administrators*.
Seit Version 4.11.3.1
- `GetMyIpByTarget(<target ip addr>)` [W]
liefert einen String mit der IP-Adresse des Interfaces, welches das Betriebssystem verwenden wird, wenn es versucht <target ip addr> zu erreichen. Diese Funktion liefert sicherer den korrekten Wert als die Verwendung der Konstante `%IPAddress%`.
Seit Version 4.11.3.1
Beispiel:

```
set $CompValue$ = getMyIpByTarget("%opsiServer%")
```

- `GetIpByName(<ip addr / ip name>)` [W]
liefert die IP-Adresse des mit <ip addr / ip name> angegebenen Rechners
Seit Version 4.11.3.2

```
set $ConstTest$ = "%IPAddress%"
    set $string1$ = "%IPAddress%"
    set $CompValue$ = getIpByName($string1$)
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif
set $CompValue$ = getIpByName("%HostID%")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $CompValue$ = getIpByName("%PCName%")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

9.3.13 (String-) Funktionen für die Lizenzverwaltung [W/L]

- `DemandLicenseKey(`poolId [, productId [, windowsSoftwareId]]`)`
Über die opsi-Webservicefunktion `getAndAssignSoftwareLicenseKey` wird vom opsi Service abgefragt, ob es für den Computer eine reservierte Lizenz gibt.

Die Datenbasis auf Grund deren die Lizenzen vergeben werden, kann die Computer ID sein, die Produkt ID oder die Windows Software ID (diese Möglichkeiten bestehen, wenn diese Vorgaben in der Lizenzkonfiguration definiert ist).

poolId, productId, windowsSoftwareId sind Strings (bzw. String-Ausdrücke). Wenn die licensePoolId nicht explizit gesetzt ist, bleibt der erste Parameter ein leerer String ". Das gilt auch für die anderen IDs – sofern diese nicht näher definiert werden.

Die Funktion gibt den Lizenzschlüssel zurück, der aus der Datenbasis ausgewählt wurde.

Beispiele:

```
set $mykey$ = DemandLicenseKey ("pool_office2007")
set $mykey$ = DemandLicenseKey ("", "office2007")
set $mykey$ = DemandLicenseKey ("", "", "{3248FOA8-6813-11D6-A77B}")
```

- `FreeLicense(`poolId [, productId [,windowsSoftwareId]])``
Über die Funktion `freeSoftwareLicense` des opsi Services wird die aktuell belegte Lizenz frei gegeben. Diese Syntax ist analog zum Syntax `DemandLicenseKey` zu sehen: Beispiel:

```
DefVar $opsireult$
set $opsireult$ = FreeLicense("pool_office2007")
```

`$opsireult$` wird zu einem leeren String umgesetzt, wenn kein Fehler auftritt und wenn eine Fehler auftritt, wird der Fehlertext ausgegeben.

9.3.14 Abrufen der Fehlerinformationen von Serviceaufrufen [W/L]

- `getLastServiceErrorClass`
liefert einen String zurück, welcher den Namen der Fehlerklasse des letzten Serviceaufrufs zurück. Wenn der letzte Serviceaufruf keine Fehlermeldung verursacht hat, gibt die Funktion den Wert "None" zurück.
- `getLastServiceErrorMessage`
liefert einen String zurück, welcher die Fehlermeldung des letzten Serviceaufrufs entspricht. Wenn der letzte Serviceaufruf keine Fehlermeldung verursacht hat, gibt die Funktion den Wert "None" zurück.

Da die Nachrichtenstrings sich immer mal wieder ändern, wird für die Logik des Grundskriptes die Verwendung des Klassennamen empfohlen.

Beispiel:

```
if getLastServiceErrorClass = "None"
    comment "kein Fehler aufgetreten"
endif
```

9.4 String-Listenverarbeitung [W/L]

Eine String-Liste (oder ein String-Listenwert) ist ein Sequenz eines String-Wertes. Für diese Werte gibt es die Variable der String-Listen. Sie sind wie folgt definiert

```
DefStringList <VarName>
```

Ein String-Listenwert ist einer String-Listenvariable zugeteilt:

```
Set <VarName> = <StringListValue>
```

String-Listen können auf vielfältige Weise erzeugt bzw. „eingefangen“ werden. Sie werden in String-Listen-Ausdrücken verarbeitet. Der einfachste String-Listen-Ausdruck ist das Setzen eines (String-Listen-) Wertes auf eine (String-Listen-) Variable.

Für die folgenden Beispiele sei generell eine String-Listen-Variable `$list1$` definiert:

DefStringList \$list1\$

Diese Variable lässt sich auf ganz unterschiedliche Weise mit Inhalten füllen: Wenn wir Variablen mit `<String0>`, `<StringVal>`, .. benennen bedeutet das, dass diese für jeden beliebigen String-Ausdruck stehen können.

Wir beginnen mit einer speziellen und sehr hilfreichen Art von String-Listen: Funktionen – also aufgerufene Hashes oder zugehörige Arrays – welche aus einer Zeile von dem Aufruf `KEY=VALUE` stammen. Tatsache ist, dass jede Funktion eine Funktion ermitteln sollte, welche einen VALUE mit einem KEY assoziiert. Jeder KEY sollte in dem ersten Abschnitt einer Zeile auftreten (während verschiedene KEYS mit identischen VALUE verbunden sein können).

9.4.1 Info Maps

- `getHWBiosInfoMap //since 4.11.4 [L/W]`
Liefert Infos zur Hardware aus dem BIOS Keys sind (mit Beispiel Werten):

```

bios.Vendor=Award Software International, Inc.
bios.Version=F9b
bios.Start Segment=E000
bios.ReleaseDate=07/08/2010
bios.RomSize=1024 k
sysinfo.Manufacturer=Gigabyte Technology Co., Ltd.
sysinfo.Product Name=GA-MA78GM-UD2H
sysinfo.Version=
sysinfo.Serial Number=
sysinfo.UUID=303032343144323730434336FFFFFFFF
sysinfo.SKU Number=
sysinfo.Family=
board.Manufacturer=Gigabyte Technology Co., Ltd.
board.Product=GA-MA78GM-UD2H
board.Version=x.x
board.Serial Number=
board.Asset Tag=
board.Feature Flags=01101001
board.Location in Chassis=
board.Chassis Handle=6261
board.Board Type=79 Unknown
board.Number of Contained Object Handles=116
enclosure.Manufacturer=Gigabyte Technology Co., Ltd.
enclosure.Version=
enclosure.Serial Number=
enclosure.Asset Tag Number=
enclosure.Type=Desktop
enclosure.Power Supply State=Unknown
enclosure.BootUp State=Unknown

```

- `getLinuxVersionMap //since 4.11.4 [L]`
Keys sind (mit Beispiel Werten):

```

Distributor ID=Ubuntu
Description=Ubuntu 12.04.2 LTS
Release=12.04
Codename=precise
kernel name=Linux
node name=detlefv05
kernel release=3.2.0-40-generic-pae
kernel version=#64-Ubuntu SMP Mon Mar 25 21:44:41 UTC 2013
machine=i686
processor=athlon
hardware platform=i386
operating system=GNU/Linux
SubRelease

```

- `getMSVersionMap` [W]
fragt die Betriebssysteminformationen lokal ab und schreibt die Informationen in eine String-Liste.

Im Moment existieren folgende Schlüssel

- `major_version`
- `minor_version`
- `build_number`
- `platform_id`
- `csd_version`
- `service_pack_major`
- `service_pack_minor`
- `suite_mask`
- `product_type_nr`
- `2003r2`
- `ReleaseID`
- `prodInfoText`
- `prodInfoNumber`

Die Ergebnisse von `suite_mask` und `product_type_nr` sind Zahlen, die aus bitweisen-or-Verknüpfungen der folgenden Werte gebildet sein können.

`product_type_nr`

```
0x00000001 (VER_NT_WORKSTATION)
0x00000002 (VER_NT_DOMAIN_CONTROLLER)
0x00000003 (VER_NT_SERVER)
```

`SuiteMask`

```
0x00000001 (VER_SUITE_SMALLBUSINESS)
0x00000002 (VER_SUITE_ENTERPRISE)
0x00000004 (VER_SUITE_BACKOFFICE)
0x00000008 (VER_SUITE_COMMUNICATIONS)
0x00000010 (VER_SUITE_TERMINAL)
0x00000020 (VER_SUITE_SMALLBUSINESS_RESTRICTED)
0x00000040 (VER_SUITE_EMBEDDEDNT)
0x00000080 (VER_SUITE_DATACENTER)
0x00000100 (VER_SUITE_SINGLEUSERTS)
0x00000200 (VER_SUITE_PERSONAL)
0x00000400 (VER_SUITE_SERVERAPPLIANCE)
```

- `ReleaseID`. Der dazugehörige Wert gibt die Release von *Windows 10* an wie z.B. *1511*.
Leerstring wenn nicht vorhanden.
Der Wert kommt aus der Registry: "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion" "ReleaseID"

- `prodInfoText`.Der dazugehörige Wert gibt die verwendeten Windows Edition als String an wie z.B. *PRODUCT_PROFESSIONAL*.
- `prodInfoNumber`.Der dazugehörige Wert gibt die verwendeten Windows Edition als Zahl an wie z.B. *48*.

ProdInfoNumber und ProdInfoText

DecNum	HexNum	Text
00	00	An unknown product
01	01	Ultimate Edition"
02	02	Home Basic Edition
03	03	Home Premium Edition
04	04	Enterprise Edition
05	05	Home Basic Edition
06	06	Business Edition
07	07	Server Standard Edition (full installation)
08	08	Server Datacenter Edition (full installation)
09	09	Small Business Server
10	0A	Server Enterprise Edition (full installation)
11	0B	Starter Edition
12	0C	Server Datacenter Edition (core installation)
13	0D	Server Standard Edition (core installation)
14	0E	Server Enterprise Edition (core installation)
15	0F	Server Enterprise Edition for Itanium-based Systems
16	10	Business Edition
17	11	Web Server Edition (full installation)
18	12	Cluster Server Edition
19	13	Home Server Edition
20	14	Storage Server Express Edition
21	15	Storage Server Standard Edition
22	16	Storage Server Workgroup Edition
23	17	Storage Server Enterprise Edition
24	18	Server for Small Business Edition
25	19	Small Business Server Premium Edition
26	1A	PRODUCT_HOME_PREMIUM_N
27	1B	PRODUCT_ENTERPRISE_N
28	1C	PRODUCT_ULTIMATE_N
29	1D	PRODUCT_WEB_SERVER_CORE
30	1E	Windows Essential Business Server Management Server
31	1F	Windows Essential Business Server Security Server
32	20	Windows Essential Business Server Messaging Server
33	21	Server Foundation
34	22	PRODUCT_HOME_PREMIUM_SERVER
35	23	PRODUCT_SERVER_FOR_SMALLBUSINESS_V
36	24	Server Standard Edition without Hyper-V (full installation)
37	25	Server Datacenter Edition without Hyper-V (full installation)
38	26	Server Enterprise Edition without Hyper-V (full installation)
39	27	Server Datacenter Edition without Hyper-V (core installation)
40	28	Server Standard Edition without Hyper-V (core installation)
41	29	Server Enterprise Edition without Hyper-V (core installation)
48	30	PRODUCT_PROFESSIONAL
49	31	PRODUCT_PROFESSIONAL_N
50	32	PRODUCT_SB_SOLUTION_SERVER
51	33	PRODUCT_SERVER_FOR_SB_SOLUTIONS
52	34	PRODUCT_STANDARD_SERVER_SOLUTIONS
53	35	PRODUCT_STANDARD_SERVER_SOLUTIONS_CORE
54	36	PRODUCT_SB_SOLUTION_SERVER_EM

55	37	PRODUCT_SERVER_FOR_SB_SOLUTIONS_EM
56	38	PRODUCT_SOLUTION_EMBEDDEDSEVER
57	39	PRODUCT_SOLUTION_EMBEDDEDSEVER_CORE
59	3B	PRODUCT_ESSENTIALBUSINESS_SERVER_MGMT
60	3C	PRODUCT_ESSENTIALBUSINESS_SERVER_ADDL
61	3D	PRODUCT_ESSENTIALBUSINESS_SERVER_MGMTSVC
62	3E	PRODUCT_ESSENTIALBUSINESS_SERVER_ADDLSVC
63	3F	PRODUCT_SMALLBUSINESS_SERVER_PREMIUM_CORE
64	40	PRODUCT_CLUSTER_SERVER_V
65	41	PRODUCT_EMBEDDED
66	42	PRODUCT_STARTER_E
67	43	PRODUCT_HOME_BASIC_E
68	44	PRODUCT_HOME_PREMIUM_E
69	45	PRODUCT_PROFESSIONAL_E
70	46	PRODUCT_ENTERPRISE_E
71	47	PRODUCT_ULTIMATE_E
72	48	PRODUCT_ENTERPRISE_EVALUATION
84	54	PRODUCT_ENTERPRISE_N_EVALUATION
98	62	PRODUCT_CORE_N
99	63	PRODUCT_CORE_COUNTRYSPECIFIC
100	64	PRODUCT_CORE_SINGLELANGUAGE
101	65	PRODUCT_CORE
121	79	PRODUCT_EDUCATION
122	7A	PRODUCT_EDUCATION_N
125	7D	Windows Enterprise 2015 LTSB
126	7E	Windows Enterprise 2015 LTSB N
129	81	Windows Enterprise 2015 LTSB Evaluation
130	82	Windows Enterprise 2015 LTSB N Evaluation

Beispiel:
Der Code

```
DefStringList $INST_Resultlist$
DefStringList $INST_Resultlist2$

message "getMSVersionMap"
comment "get value by winst function"
set $INST_Resultlist$ = getMSVersionMap
```

Liefert z.B. im Log:

```
message getMSVersionMap
comment: get value by winst function

Set $INST_Resultlist$ = getMSVersionMap
  retrieving strings from getMSVersionMap [switch to loglevel 7 for debugging]
  (string 0)major_version=5
  (string 1)minor_version=1
  (string 2)build_number=2600
  (string 3)platform_id=2
  (string 4)csd_version=Service Pack 3
  (string 5)service_pack_major=3
  (string 6)service_pack_minor=0
  (string 7)suite_mask=256
  (string 8)product_type_nr=1
  (string 9)2003r2=false
```

Anmerkung

Background infos for getMSVersionMap

- <http://msdn.microsoft.com/en-us/library/ms724385%28VS.85%29.aspx>
- <http://msdn.microsoft.com/en-us/library/dd419805.aspx>
- <http://msdn.microsoft.com/en-us/library/ms724833%28VS.85%29.aspx>

- getFileInfoMap(<file name>) : stringlist [W]
- getFileInfoMap32(<file name>) : stringlist //since 4.11.6.6 [W]
- getFileInfoMap64(<file name>) : stringlist //since 4.11.6.6 [W]
- getFileInfoMapSynative(<file name>) : stringlist //since 4.11.6.6 [W]

findet die Versionsinformationen, die im FILENAME verborgen sind und schreibt sie in eine String-Listen Funktion. Zur Zeit existieren folgende Schlüssel,

- Comments
- CompanyName
- FileDescription
- FileVersion
- InternalName
- LegalCopyright
- LegalTrademarks
- OriginalFilename
- PrivateBuild
- ProductName
- ProductVersion
- SpecialBuild
- Language name <index>
- Language ID <index>
- file version with dots
- file version
- product version

Verwendung: Wenn wir folgende definieren und aufrufen

```
DefStringList FileInfo
DefVar $InterestingFile$
Set $InterestingFile$ = "c:\program files\my program.exe"
set FileInfo = getFileInfoMap($InterestingFile$)
```

bekommen wir die Werte, die zum Schlüssel "FileVersion" dazugehören, über den Aufruf

```
DefVar $result$
set $result$ = getValue("FileVersion", FileInfo)
```

ausgegeben (für die Funktion `getValue` vgl. Abschnitt [Abschnitt 9.4.4](#)).

Beispiel:

Der Code:

```
set $InterestingFile$ = "%windir%\winst.exe"
if not (FileExists($InterestingFile$))
    set $InterestingFile$ = "%windir%\winst32.exe"
endif
set $INST_Resultlist$ = getFileInfoMap($InterestingFile$)
```

liefert z.B. im Log

```
Set $InterestingFile$ = "N:\develop\delphi\winst32\trunk\winst.exe"
The value of the variable is now: "N:\develop\delphi\winst32\trunk\winst.exe"

If
    Starting query if file exist ...
FileExists($InterestingFile$) <<< result true
not (FileExists($InterestingFile$)) <<< result false
Then
EndIf

Set $INST_Resultlist$ = getFileInfoMap($InterestingFile$)
retrieving strings from getFileInfoMap [switch to loglevel 7 for debugging]
(string 0)Language name 0=Deutsch (Deutschland)
(string 1)Language ID 0=1031
(string 2)file version=1125942857039872
(string 3)file version with dots=4.10.8.0
(string 4)product version=1125942857039872
(string 5)Comments=
(string 6)CompanyName=uib gmbh (www.uib.de)
(string 7)FileDescription=opsi.org
(string 8)FileVersion=4.10.8.0
(string 9)InternalName=
(string 10)LegalCopyright=uib gmbh under GPL
(string 11)LegalTrademarks=opsi
(string 12)OriginalFilename=
(string 13)PrivateBuild=
(string 14)ProductName=opsi-winst
(string 15)ProductVersion=4.0
(string 16)SpecialBuild=
```

- `getLocaleInfoMap [W]`
fragt die Systeminformationen lokal ab und schreibt die Informationen in eine String-Liste.

Im Moment existieren folgende Schlüssel

- `language_id_2chars` (eine „Zwei-Buchstaben“ Namensangabe der default Systemsprache)
- `language_id` (eine „Drei-Buchstaben“ Namensangabe der default Systemsprache inklusive der Sprachenuntertypen)
- `localized_name_of_language`

- English_name_of_language
- abbreviated_language_name
- native_name_of_language
- country_code
- localized_name_of_country
- English_name_of_country
- abbreviated_country_name
- native_name_of_country
- default_language_id
- default_language_id_decimal
- default_country_code
- default_oem_code_page
- default_ansi_code_page
- default_mac_code_page
- system_default_language_id Hexadecimal Windows locale Id
- system_default_posix Sprache_Region (Posix Style)
- system_default_lang_region Sprache-Region (BCP 47 Style)

Die system_default Keys geben Informationen über die Sprache des installierten Betriebssystems. Die anderen Keys geben Informationen über die Lokalisierung der GUI.

Beispiel:

Der Code:

```
message "Locale Infos"
set $INST_Resultlist$ = getLocaleInfoMap
```

liefert z.B. folgendes Log:

```
message Locale Infos

Set $INST_Resultlist$ = getLocaleInfoMap
  retrieving strings from getLocaleInfoMap [switch to loglevel 7 for debugging]
    (string 0)language_id_2chars=DE
    (string 1)language_id=DEU
    (string 2)localized_name_of_language=Deutsch (Deutschland)
    (string 3)English_name_of_language=German
    (string 4)abbreviated_language_name=DEU
    (string 5)native_name_of_language=Deutsch
    (string 6)country_code=49
    (string 7)localized_name_of_country=Deutschland
    (string 8)English_name_of_country=Germany
    (string 9)abbreviated_country_name=DEU
    (string 10)native_name_of_country=Deutschland
    (string 11)default_language_id=0407
    (string 12)default_language_id_decimal=1031
    (string 13)default_country_code=49
```

```
(string 14)default_oem_code_page=850
(string 15)default_ansi_code_page=1252
(string 16)default_mac_code_page=10000
(string 17)system_default_language_id=0407
(string 18)system_default_posix=de_DE
(string 19)system_default_lang_region=de-DE
```

Verwendung: Wenn wir den Aufruf wie folgt definieren

```
DefStringList $languageInfo$
set $languageInfo$ = getLocaleInfoMap
```

bekommen wir den Wert mit dem KEY "language_id_2chars" über den Aufruf

```
DefVar $result$
set $result$ = getValue("language_id_2chars", $languageInfo$)
```

(für die Funktion `getValue` vgl. Abschnitt Abschnitt 9.4.4). Wir können nun Skripte mit folgender Konstruktion verwenden

```
if getValue("language_id_2chars", languageInfo) = "DE"
    ; installiere deutsche Version
else
    if getValue("language_id_2chars", languageInfo) = "EN"
        ; installiere englische Version
    endif
endif
```

Anmerkung

Background infos for `getLocaleInfoMap`:

- <http://msdn.microsoft.com/en-us/library/cc233968.aspx>
 - <http://msdn.microsoft.com/en-us/library/0h88fahh.aspx>
 - bcp 47 validator:
<http://schneegans.de/lv/?tags=de-de-1996&format=text>
 - <http://www.iana.org/assignments/language-subtag-registry>
 - <http://www.the-localization-tool.com/?p=698>
-

Die Funktion `GetLocaleInfoMap` ersetzt die ältere `GetLocaleInfo`, da diese Werte ausliest, die schwierig zu interpretieren sind:

- `getLocaleInfo`
(abgekündigt): Bitte `GetLocaleInfoMap` verwenden.
- `getProductMap` // since 4.11.2.4 [W/L]
liefert eine info map über das opsi product welches gerade installiert wird.
Die Funktion arbeitet nur korrekt, wenn *opsi-winst/opsi-script* im opsi service mode aufgerufen wird.
keys sind: id, name, description, advice, productversion, packageversion, priority, installationstate, lastactionrequest, lastactionresult, installedversion, installedpackage, installedmodificationtime, actionrequest

Beispiel:

```
set $INST_Resultlist$ = getProductMap
set $string1$ = getValue("id", $INST_Resultlist$)
```

liefert z.B. folgenden log:

```
Set $INST_Resultlist$ = getProductMap
  retrieving strings from getProductMap [switch to loglevel 7 for debugging]
  (string 0)id=opsi-script-test
  (string 1)name=opsi-winst test
  (string 2)description=Test and example script for opsi-winst
  (string 3)advice=
  (string 4)productversion=4.11.2
  (string 5)packageversion=1
  (string 6)priority=0
  (string 7)installationstate=unknown
  (string 8)lastactionrequest=setup
  (string 9)lastactionresult=successful
  (string 10)installedversion=4.11.2
  (string 11)installedpackage=1
  (string 12)installedmodificationtime=
  (string 13)actionrequest=setup

Set $string1$ = getValue("id", $INST_Resultlist$)
  retrieving strings from $INST_Resultlist$ [switch to loglevel 7 for debugging]
  (string 0)id=opsi-script-test
  (string 1)name=opsi-winst test
  (string 2)description=Test and example script for opsi-winst
  (string 3)advice=
  (string 4)productversion=4.11.2
  (string 5)packageversion=1
  (string 6)priority=0
  (string 7)installationstate=unknown
  (string 8)lastactionrequest=setup
  (string 9)lastactionresult=successful
  (string 10)installedversion=4.11.2
  (string 11)installedpackage=1
  (string 12)installedmodificationtime=
  (string 13)actionrequest=setup
```

The value of the variable "\$string1\$" is now: "opsi-script-test"

9.4.2 Erzeugung von String-Listen aus vorgegebenen String-Werten [W/L]

- `createStringList` (Stringwert0, Stringwert1 ,... `) [W/L]`
erzeugt eine neue Liste aus den aufgeführten einzelnen String-Werten, z.B. liefert

```
set $list1$ = createStringList ('a','b', 'c', 'd')
```

die ersten vier Buchstaben des Alphabets.

- `splitString` (Stringwert1, Stringwert2)` [W/L]`
erzeugt die Liste der Teilstrings von String-Wert1, die jeweils durch String-Wert2 voneinander getrennt sind. Z.B. bildet

```
set $list1$ = splitString ("\\server\share\directory", "\")
```

die Liste

```
"", "", "server", "share", "directory"
```

Ist der übergebene String confidential, so werden es die hier entstehenden Teile auch sein.

- `splitStringOnWhiteSpace (<string>)` [W/L]
zerlegt <string> in die durch "leere" Zwischenräume definierten Abschnitte. Das heißt z.B.

```
set $list1$ = splitStringOnWhiteSpace("Status Lokal Remote Netzwerk")
```

liefert die Liste

```
"Status", "Lokal", "Remote", "Netzwerk"
```

unabhängig davon, wie viele Leerzeichen oder ggf. Tabulatorzeichen zwischen den "Wörtern" stehen.

Ist der übergebene String confidential, so werden es die hier entstehenden Teile auch sein.

9.4.3 Laden der Zeilen einer Textdatei in eine String-Liste

- `loadTextFile (`Dateiname`)` [W/L]
liest die Zeilen der Datei des (als String) angegebenen Namens ein und generiert aus ihnen eine String-Liste.
- `LoadTextFileWithEncoding(<file name> , <encoding>)` [W/L]
Liefert eine Stringliste mit dem Inhalt von <file name> welcher beim Einlesen von <encoding> in das laufende Encoding des Systems konvertiert wurde.
- `loadUnicodeTextFile (`Dateiname`)` [W]
liest die Zeilen der Unicode-Datei des (als String) angegebenen Namens ein und generiert aus ihnen eine String-Liste.
Die Strings in der String-Liste werden dabei (gemäß den Betriebssystem-Defaults) in 8 Bit-Code konvertiert.
- `getSectionNames (`Dateiname`)` [W/L]
liest die ausgewählte Datei als eine ini-Datei aus, durchsucht alle Zeilen nach dem Begriff [*<SectionName>*] und gibt die einfachen Sektionsnamen (ohne Klammern) aus.

9.4.4 (Wieder-) Gewinnen von Einzelstrings aus String-Listen oder Dateien [W/L]

- `composeString (<StringListe>, <LinkString>)` [W/L]
Mit dieser Funktion lässt sich die Zerlegung eines Strings in einer String-Liste z.B. nach vorheriger Transformation (s. den Abschnitt „Transformation von String-Listen“) – rückgängig machen.
Zum Beispiel:
Wenn \$list1\$ für die Liste *a, b, c, d, e* steht, erhält die String-Variable line mittels `$line$ = composeString ($list1$, " | ")` den Wert *"a / b / c / d / e"*.
- `takeString(<index>,<list>)` [W/L]
liefert aus der String-Liste <list> den String mit dem Index <index>
zum Beispiel liefert (wenn \$list1\$ wie eben die Liste der ersten fünf Buchstaben des Alphabets ist) `takeString (2, list1)` den String *c* (der Index beruht auf einer mit 0 beginnenden Nummerierung der Listenelemente).
Negative Werte des index werden die Werte abwärts der Liste ausgelesen. Z.B., `takeString (-1, list1)` gibt das letzte Listenelement zurück; das ist *e*.

- `takeFirstStringContaining(<list>,<search string>)` [W/L]
Liefert den ersten String einer Liste der `<search string>` enthält. Liefert einen leeren String wenn kein passender String gefunden wird.
- `getValue (<key>, <list>)` [W/L]
Diese Funktion versucht eine String-Liste `<list>` als eine Liste aus Zeilen der Form
`key=value`
auszulesen.
Dazu sucht die Funktion die erste Zeile, wo der String-Key vor dem Gleichheitszeichen erscheint und gibt den Rest der Zeile zurück (der String, der nach dem Gleichheitszeichen folgt). Wenn es keine passende Zeile gibt, wird der Wert `NULL` zurückgegeben.
Die Funktion ist z.B. notwendig für die Nutzung von `getLocaleInfoMap` und `getFileVersionMap` String list Funktionen (vgl. Abschnitt Abschnitt 9.4.1).
- `getValueBySeparator(<key string>,<separator string>,<hash string list>)` //since 4.11.2.1 [W/L]
arbeitet wie `getValue` aber der Trenner zwischen `key` und `value` (`<separator string>`) muss angegeben werden so das mit maps gearbeitet werden kann wie
`key:value`
- `getValueFromFile(<key string>, <file name>)` //since 4.11.4.4 [W/L]
Sucht in `<file name>` nach einem key/value Paar mit dem key `<key string>` und separator string `=` und liefert den gefundenen value. Wenn `<key string>` nicht gefunden wird liefert die Funktion einen leeren string.
- `getValueFromFileBySeparator(<key string>,<separator string>, <file name>)` //since 4.11.4.4 [W/L]
Sucht in `<file name>` nach einem key/value Paar mit dem key `<key string>` und separator string `<separator string>` und liefert den gefundenen value. Wenn `<key string>` nicht gefunden wird liefert die Funktion einen leeren string.

Die pseudo-integer Funktion `* count (<list>)` [W/L]

ist eine pseudo Integer Funktion. Sie zählt die Elemente einer String-Liste `<list>`; das Resultat wird in einen String gewandelt. Ist `$list1$` z.B.

`a, b, c, d, e`

so hat `count ($list1$)` den Wert "5".

9.4.5 String-Listen-Erzeugung mit Hilfe von Sektionsaufrufen

- `retrieveSection (`Sektionsname`)` [W/L]
gibt die Zeilen einer aufgerufenen Sektion aus.
- `getOutputStreamFromSection (`Sectionname`)` [W/L]
„fängt“ – derzeit bei `DosInAnIcon` (`ShellInAnIcon`), `ExecWith` und `ExecPython` Aufrufen – die Ausgabe der Kommandozeilenprogramme in der Form einer String-Liste ein. Z.B. liefert der Ausdruck `getOutputStreamFromSection ('DosInAnIcon_netstart')`
wenn die aufgerufene Sektion definiert ist durch

```
set $list$ = getOutputStreamFromSection ('DosInAnIcon_netstart')
```

```
[DosInAnIcon_netstart]
net start
```

eine Reihe von Zeilen, die u.a. die Auflistung aller laufenden Dienste enthalten und dann weiterbearbeitet werden können.

Seit 4.11.4.2 gibt es für einfache Befehle und besonders (aber nicht nur) für den Einsatz unter Linux folgende *Abkürzung*, welche unter Windows im *sysnative* mode läuft:

- `shellCall (<command string>) [W/L]`

```
set $list$= shellCall('net start')
```

ist eine Abkürzung von:

```
set $list$ = getOutputStreamFromSection ('DosInAnIcon_netstart winst /sysnative')
```

```
[DosInAnIcon_netstart]
net start
```

- `getReturnListFromSection (`Sectionname`) [W/L]`
In Sektionen bestimmter Typen – derzeit implementiert nur für XMLPatch- und opsiServiceCall-Sektionen – existiert eine spezifische Return-Anweisung, die ein Ergebnis der Sektion als String-Liste zur Verfügung stellt.

XMLPatch Beispiel:

Die Anweisung

```
set $list1$ =getReturnListFromSection ('XMLPatch_mime "c:\mimetypes.rdf"')
```

liefert eine spezifisch selektierte Liste von Knoten der XML-Datei mimetypes.rdf liefern. Näheres zu XMLPatch-Sektionen ist der Dokumentation im Kapitel Abschnitt 10.7 zu entnehmen.

OpsiServiceCall Beispiel:

```
DefStringList $result$
Set $result$=getReturnListFromSection("opiservicecall_clientIdsList")

[opiservicecall_clientIdsList]
"method": "getClientIds_list"
"params": []
```

9.4.6 String-Listen aus der Registry [W]

- `getRegistryKeyList32(<regkey>)`
Liefert eine Liste mit dem Namen aller Keys direkt unterhalb von <regkey>. 32 Bit Modus (mit Redirection). Seit 4.11.3
- `getRegistryKeyList64(<regkey>)`
Liefert eine Liste mit dem Namen aller Keys direkt unterhalb von <regkey>. 64 Bit Modus (ohne Redirection). Seit 4.11.3
- `getRegistryKeyListSysnative(<regkey>)`
Liefert eine Liste mit dem Namen aller Keys direkt unterhalb von <regkey>. Modus abhängig von der Architektur des Betriebssystems. Seit 4.11.3
- `getRegistryVarList32(<regkey>)`
Liefert eine Liste mit dem Namen aller Werte direkt unterhalb von <regkey>. 32 Bit Modus (mit Redirection). Seit 4.11.3

- `getRegistryVarList64(<regkey>)`
Liefert eine Liste mit dem Namen aller Werte direkt unterhalb von <regkey>. 64 Bit Modus (ohne Redirection). Seit 4.11.3
- `getRegistryVarListSysnative(<regkey>)`
Liefert eine Liste mit dem Namen aller Werte direkt unterhalb von <regkey>. Modus abhängig von der Architektur des Betriebssystems. Seit 4.11.3
- `getRegistryVarMap32(<regkey>)`
Liefert eine Map mit den Namen=Value Paaren aller Werte direkt unterhalb von <regkey>. 32 Bit Modus (mit Redirection). Seit 4.11.3
- `getRegistryVarMap64(<regkey>)`
Liefert eine Map mit den Namen=Value Paaren aller Werte direkt unterhalb von <regkey>. 64 Bit Modus (ohne Redirection). Seit 4.11.3
- `getRegistryVarMapSysnative(<regkey>)`
Liefert eine Map mit den Namen=Value Paaren aller Werte direkt unterhalb von <regkey>. Modus abhängig von der Architektur des Betriebssystems. Seit 4.11.3

Beispiel:

Wir erzeugen Registryeinträge mit folgender Sektion durch den Aufruf von:

```
Registry_createkeys /32Bit

openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-test]
set "var1" = "value1"
set "var2" = REG_SZ:"value2"
set "var3" = REG_EXPAND_SZ:"value3"
set "var4" = REG_DWORD:444
set "var5" = REG_BINARY:05 05 05 0F 10
set "var6" = REG_MULTI_SZ:"value6|value7|de"
openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-test\key1]
openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-test\key2]
openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-test\key3]
openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-test\key4]
```

Dann liefert uns:

```
set $list$ = getRegistryVarList32("hklm\software\opsi.org\opsi-script-test")
```

folgenden Log:

```
Set $list$ = GetRegistryVarList32("hklm\software\opsi.org\opsi-script-test")
Registry started with redirection (32 Bit)
    retrieving strings from GetRegistryVarList32 [switch to loglevel 7 for debugging]
        (string 0)var1
        (string 1)var2
        (string 2)var3
        (string 3)var4
        (string 4)var5
        (string 5)var6
```

Dann liefert uns:

```
set $list$ = getRegistryVarMap32("hkml\software\opsi.org\opsi-script-test")
```

folgenden Log:

```
Set $list$ = GetRegistryVarMap32("hkml\software\opsi.org\opsi-script-test")
retrieving strings from GetRegistryVarMap32 [switch to loglevel 7 for debugging]
  (string 0) var1=value1
  (string 1) var2=value2
  (string 2) var3=value3
  (string 3) var4=444
  (string 4) var5=05 05 05 0F 10
  (string 5) var6=value6
```

Dann liefert uns:

```
set $list$ = getRegistryKeyList32("hkml\software\opsi.org\opsi-script-test")
```

folgenden Log:

```
Set $list$ = GetRegistryKeyList32("hkml\software\opsi.org\opsi-script-test")
Registry started with redirection (32 Bit)
  retrieving strings from GetRegistryKeyList32 [switch to loglevel 7 for debugging]
    (string 1) key1
    (string 2) key2
    (string 3) key3
    (string 4) key4
```

9.4.7 String-Listen aus Produkt Properties [W/L]

- `getProductPropertyList(<propname>,<default value>)` [W/L]
Liefert eine Liste mit den aktiven Werten des multivalue Properties <propname>. Im Gegensatz zu der Funktion `GetProductProperty` welche in diesem Fall die einzelnen Werte auf einem komma separierten String zurück liefert. Diese Vorgehen wird problematisch wenn Kommas auch in der Werten vorkommen.
<default value> kann bei `getProductPropertyList` (seit 4.11.5.6) sowohl ein Stringausdruck sein welcher das erste Element der einer Liste beschreibt oder eine Liste. Diese Liste wird zurückgegeben wenn der die aktuellen Werte des Servers nicht verfügbar sind. Seit 4.11.3
Beispiel:

```
DefStringList $list$
;Property "dummymulti" has the values: ("ab", "cd", "ef", "g,h")
set $list$ = GetProductPropertyList ("dummymulti","True")
if not (" " = takeFirstStringContaining($list$,"g,h"))
  comment "GetProductPropertyList passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "GetProductPropertyList failed"
endif

set $ConstTest$ = "ab,cd,ef,g,h"
set $CompValue$ = GetProductProperty ("dummymulti","True")
if ($ConstTest$ = $CompValue$)
  comment "GetProductProperty passed"
else
  set $TestResult$ = "not o.k."
```

```

        LogWarning "GetProductProperty failed"
endif

;;;;;another Example to get a list as default-property

DefStringList $list$
DefStringList $propertyList$
Set $propertyList$ = createStringList('ab','cd','de')
Set $list$ = GetProductPropertyList ("dummymulti",$propertyList$)

```

9.4.8 Sonstige String-Listen [W/L]

- `getProfilesDirList` [W/L]
Liefert eine Liste der Pfade zu den lokalen Profilen.
[W] : Profile welche die folgenden Worte enthalten werden **nicht** berücksichtigt:
 - `localservice`
 - `networkservice`
 - `systemprofile`

Das Profil des *Default Users* ist Bestandteil der Liste.
All *User* oder *Public* sind nicht Bestandteil der Liste.

[L] : Es werden die Heimatverzeichnisse der user mit einer UID ≥ 1000 geliefert, sofern das Verzeichnis auch existiert.

Beispiel:

```
set $list1$ = getProfilesDirList
```

ergibt folgenden Log:

```

Set $list1$ = getProfilesDirList
Registry started with redirection (32 Bit)
  retrieving strings from getProfilesDirList [switch to loglevel 7 for debugging]
    (string  0)C:\Users\Administrator
    (string  1)C:\Users\Default

```

- `GetProcessList` //since 4.11.1.2; gives list of exename;pid;dom/user [W/L]
Liefert eine Liste der laufenden Prozesse.
Für jeden Prozess gibt es eine Zeile mit den folgenden ; separierten Prozessinformationen:
 - [W]: *Prozess Name*. [L]: *Kurzname des Prozesses*
 - [W/L]: *PID*
 - [W]: *Domain/User*. [L]: *User*
 - [L]: *Komplette Kommandozeile des laufenden Prozesses*

9.4.9 Transformation von String-Listen [W/L]

- `getSubList` (<Startindex> : <Endindex>, <list>) [W/L]
Liefert eine Teilliste einer vorgegebenen Liste.
Funktion:
Wenn \$list\$ z.B. für die Liste der Buchstaben *a, b, c, d, e* steht, so liefert

```
set $list1$ = getSubList(1 : 3, $list$)
```

b, c, d (Startindex und Endindex sind die Nummer des Listenelements, wenn mit 0 beginnend gezählt wird).
Defaultwert des Startindex ist 0, des Endindex der letzte Index der Liste. Z.B. ergibt mit obiger Festlegung für `$list$`

```
set $list1$ = getSubList(1 : , $list$)
```

b, c, d, e.

```
set $list1$ = getSubList(:, $list$)
```

ist genau eine Kopie der ursprünglichen Liste.

Es besteht die Möglichkeit den Endindex mit Rückwärtszählung zu bestimmen:

```
set $list1$ = getSubList(1 : -1, $list$)
```

ist die Teilliste der Elemente vom 1. bis zum letzten Element der ursprünglichen Liste – im obigen Beispiel also wieder *b, c, d, e.*

```
set $list1$ = getSubList(1 : -2, $list$)
```

ist die Teilliste der Elemente vom 1. bis zum vorletzten Element der ursprünglichen Liste – im obigen Beispiel also wieder *b, c, d.*

- `getListContaining(<list>,<search string>)` [W/L]
Liefert eine Teilliste mit allen Strings welche den `<search string>` enthalten.
- `getListContainingList(<list1>,<list2>)` //since 4.11.3.7 [W/L]
Liefert die Schnittmenge von `list1` und `list2`
- `getSubListByMatch (<search string>, <target list>)` :stringlist //since 4.12.0.14 [W/L]
Liefert eine Teilliste von `<target list>` mit allen Strings welche mit dem `<search string>` identisch sind.
Die Überprüfung auf Gleichheit ist nicht Case Sensitive.
- `getSubListByMatch (<search list>, <target list>)` :stringlist //since 4.12.0.14 [W/L]
Liefert eine Teilliste von `<target list>` mit allen Strings welche mit einem der Strings aus `<search list>` identisch sind.
Die Überprüfung auf Gleichheit ist nicht Case Sensitive.
- `getSubListByContaining (<search string>, <target list>)` :stringlist //since 4.12.0.14 [W/L]
Liefert eine Teilliste von `<target list>` mit allen Strings in welchen `<search string>` vorkommt.
Die Überprüfung auf Gleichheit ist nicht Case Sensitive.
- `getSubListByContaining (<search list>, <target list>)` :stringlist //since 4.12.0.14 [W/L]
Liefert eine Teilliste von `<target list>` mit allen Strings in denen einer der Strings aus `<search list>` vorkommt.
Die Überprüfung auf Gleichheit ist nicht Case Sensitive.
- `getSubListByKey (<search string>, <target list>)` :stringlist //since 4.12.0.14 [W/L]
Liefert eine Teilliste von `<target list>` mit allen Strings welche mit `<search string>` anfangen.
Die Überprüfung auf Gleichheit ist nicht Case Sensitive.

- **getSubListByKey** (<search list>, <target list>) :stringlist //since 4.12.0.14 [W/L]
Liefert eine Teilliste von der key/value paar liste <target list> mit allen Strings, deren key in <search list> vorkommt.
Die Überprüfung auf Gleichheit ist nicht Case Sensitive.
- **getKeyList** (<list>) :stringlist //since 4.12.0.14 [W/L]
Liefert von einer Liste von key/value paaren im Format key=value die Liste aller keys. Ist ein eintrag in <list> nicht im format key=value, so wird der komplette string zurückgeliefert.
- **takeFirstStringContaining**(<list>,<search string>) [W/L]
Liefert den ersten String von <list> welcher den <search string> enthält.
Liefert einen Leerstring wenn <search string> nicht gefunden wird.
- **addtolist**(<list>,<string>) [W/L]
Hängt den String <string> an die Liste <list> an.
- **addlisttolist**(<list1>,<list2>) [W/L]
Hängt die Liste <list2> an die Liste <list1> an.
- **reverse** (<list>) [W/L]
kehrt die Reihenfolge der Aufzählung um – aus *a, b, c, d, e* wird mit

```
set $list1$ = reverse ($list$)
```

also *e, d, c, b, a*.

- **emptylist** (<list>) //since 4.11.3.7 [W/L]
Leert die Liste.
- **reencodestrlist**(<list>, <from>, <to>) //since 4.11.4.2 [W/L]
liefert die Stringliste <list> mit dem encoding <to> zurück wobei davon ausgegangen wird das <list> gemäß <from> encoded war. <from> und <to> sind dabei encodings wie sie im Kapitel Abschnitt [6.3](#) aufgelistet sind.
- **removeFromListByContaining**(<search string>, <target list>) : stringlist //since 4.11.5.1 [W/L]
Liefert eine Kopie von <target list> bei der alle Zeilen entfernt sind in denen <search string> vorkommt. Der Vergleich ist case sensitiv.
- **removeFromListByContaining**(<search list>, <target list>) : stringlist [W/L]
Liefert eine Kopie von <target list> bei der alle Zeilen entfernt sind in denen ein String aus <search list> vorkommt. Der Vergleich ist case sensitiv.

Examples:

File "%Scriptpath%\test-files\encoding\10lines.txt" is:

```

line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
line 10

```

Code from opsi-script-test:

```

comment ""
comment "-----"
comment "Testing: "
message "removeFromListByContaining"
set $string1$ = "%Scriptpath%\test-files\encoding\10lines.txt"
set $list1$ = loadTextFileWithEncoding($string1$, "cp1252")
comment "search with string"
comment "search with string constant"
set $ConstTest$ = "9"
set $list2$ = removeFromListByContaining("line 5", $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $ConstTest$ = "9"
set $list2$ = removeFromListByContaining("LINE 5", $list1$)
comment "the match is case insensitive"
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $ConstTest$ = "0"
set $list2$ = removeFromListByContaining("line", $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $ConstTest$ = "8"
comment "searchstr 1 will found in 'line 1' and 'line 10'"
set $list2$ = removeFromListByContaining("1", $list1$)
set $CompValue$ = count($list2$)

```

```
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

comment "search with string function"
set $ConstTest$ = "9"
set $list2$ = removeFromListByContaining(trim(" line 5 "), $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

comment "search with string variable"
set $string1$ = "line 5"
set $ConstTest$ = "9"
set $list2$ = removeFromListByContaining($string1$, $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

comment "search with string list"
comment "search with string list variable"
set $list3$ = createStringList ('1', '2', '3', '4', '5')
comment "searchstr 1 will found in 'line 1' and 'line 10'"
set $ConstTest$ = "4"
set $list2$ = removeFromListByContaining($list3$, $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

comment "search with string list variable"
comment "searchstr 1 will found in 'line 1' and 'line 10'"
set $ConstTest$ = "4"
set $list2$ = removeFromListByContaining(createStringList ('1', '2', '3', '4', '5'), $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

- `removeFromListByMatch(<searchstring>,<target list>)` : `stringlist` //since 4.11.6 [W/L]
Liefert eine Kopie von <target list> bei der alle Zeilen entfernt sind die gleich <search string> sind. Der Vergleich ist case sensitiv.
- `setStringInListAtIndex(<newstring>,<list>,<indexstr>)` : `stringlist` //since 4.11.6 [W/L]
Liefert eine Kopie von <list> bei dem der String an <index> durch <newstring> ersetzt ist. Wenn <index> zu groß ist wird <newstring> angehängt.
Im Fehlerfall ist das Ergebnis eine leere Liste.

Replaces in the existing stringlist <list> the existing string at <index> by <newstring>

9.4.10 Iteration durch String-Listen [W/L]

Eine besonders wichtige Anwendung von String-Listen beruht auf der Möglichkeit, die Elemente einer String-Liste zu durchlaufen und für jedes Element eine vorgegebenes Anweisungsschema auszuführen:

Die Syntax für eine solche Iteration („Wiederholungsanweisung“) lautet:

- `for %s% in <list> do <eine Anweisung>`

Dabei wird %s% durch diesen Ausdruck und nur für diese Stelle als String-Variable deklariert und ist danach wieder unbekannt. Innerhalb der Anweisung wird jedes Vorkommen von %s% (oder wie auch immer eine entsprechende Variable benannt ist) der Reihe nach durch die verschiedenen Elemente der Liste ersetzt.



Achtung

Die Ersetzung ist (wie bei Systemkonstanten) rein textuell, d.h. genau die Zeichenfolge %s% wird z.B. durch die Werte a b c - ersetzt. Sind die Strings a,b,c gemeint, muss in der auszuführenden Anweisung %s% von Anführungszeichen eingeschlossen sein.

Ein Beispiel: Wenn \$list1\$ für a, b, c, d, e steht und \$line\$ als String-Variable deklariert ist, so bedeutet:

```
for %s% in $list1$ do    set $line$ = $line$ + "%s%"
```

der Reihe nach

```
$line$ = $line$ + "a"
$line$ = $line$ + "b"
$line$ = $line$ + "c"
$line$ = $line$ + "d"
$line$ = $line$ + "e"
```

so dass am Ende \$line\$ den Wert *abcde* trägt. Wenn wir die einfachen Anführungszeichen um das %s% weglassen würden, bekämen wir bei jedem Schritt der Iteration einen Syntaxfehler gemeldet.

9.5 opservicecall und JSON Funktionen [W/L]

Diese Funktionen dienen dazu JSON Ausdrücke auf String oder Stringlisten zu analysieren und zu modifizieren. Notwendig ist hierfür, die Struktur des zu bearbeitenden JSON-Ausdrucks zu kennen und zu verstehen. Dies kann dazu verwendet werden, um über opservicecall Sektionen mit den objektorientierten opsi-webservice Methoden zu arbeiten.

- `jsonIsValid(<jsonstr>)` : `boolean` //since 4.11.6: [W/L]
Liefert `true` zurück wenn, <jsonstr> einen gültigen JSON Ausdruck enthält.
- `jsonIsArray(<jsonstr>)` : `boolean` //since 4.11.6: [W/L]
Liefert `true` zurück wenn, <jsonstr> ein gültiges JSON Array enthält.
- `jsonIsObject(<jsonstr>)` : `boolean` //since 4.11.6: [W/L]
Liefert `true` zurück wenn, <jsonstr> ein gültiges JSON Object enthält.
- `jsonAsObjectHasKey(<jsonstr>,<keystr>)` : `boolean` //since 4.11.6: [W/L]
Liefert `true` zurück wenn, <jsonstr> ein gültiges JSON Object welches <keystr> als key enthält.
Folgendes Beispiel würde `true` zurückliefern:

```
jsonAsObjectHasKey('{ "productVersion" : "4.4.1", "packageVersion" : "2", "productId" : "jedit" }, 'productId')
```

- `jsonAsArrayCountElements(<jsonstr>)` : `intstr` //since 4.11.6: [W/L]
Wenn <jsonstr> ein gültiges JSON Array enthält ist der Rückgabewert ein String mit der Zahl der Elemente des Arrays.
Im Fehlerfall = `"0"`
- `jsonAsObjectCountElements(<jsonstr>)` : `intstr` //since 4.11.6: [W/L]
- `jsonAsArrayGetElementByIndex(<jsonstr>, <indexstr>)` : `jsonstring` //since 4.11.6: [W/L]
Liefert vom JSON Array <jsonstr> das Element mit dem Index <indexstr>
Der Index beginnt bei 0.
Im Fehlerfall = `""`
- `jsonAsObjectGetValueByKey(<jsonstr>, <keystr>)` : `valuestring` //since 4.11.6: [W/L]
Liefert vom JSON Object <jsonstr> den Wert des Key <keystr>
Im Fehlerfall = `""`
- `jsonAsObjectSetValueByKey(<jsonstr>, <keystr>,<valuestring>)` : `jsonstring` //since 4.11.6: [W/L]
Liefert einen String mit dem in <jsonstr> übergebenen JSON Object bei dem für den Key <keystr> der Wert <valuestring> gesetzt ist. Ist der Key nicht vorhanden, so wird er erzeugt.
Wenn <valuestring> als Stringwert erzeugt werden soll (also in doppelten Anführungszeichen), dann verwenden Sie besser die folgende Funktion: `jsonAsObjectSetStringtypeValueByKey`.
Im Fehlerfall = `""`
- `jsonAsObjectSetStringtypeValueByKey(<jsonstr>, <keystr>,<valuestring>)` : `jsonstring` //since 4.11.6: [W/L]
Liefert einen String mit dem in <jsonstr> übergebenen JSON Object bei dem für den Key <keystr> der Wert <valuestring> als String (also in doppelten Anführungszeichen) gesetzt ist. Ist der Key nicht vorhanden, so wird er erzeugt.
Wenn <valuestring> nicht als Stringwert erzeugt werden soll , dann verwenden Sie besser die vorherige Funktion: `jsonAsObjectSetValueByKey`.
Im Fehlerfall = `""`

- `jsonAsObjectDeleteByKey(<jsonstr>, <keystr>)` : `jsonstring` //since 4.11.6.4: [W/L]
Liefert einen String mit dem in <jsonstr> übergebenen JSON Object bei dem das key-value Paar mit dem Key <keystr> entfernt wurde.
- `jsonAsArrayPutObjectByIndex(<jsonstr>, <indexstr>, <objectstr>)` : `jsonstring` //since 4.11.6: [W/L]
Liefert einen String mit dem in <jsonstr> übergebenen JSON Array bei dem am Index <indexstr> das Object <objectstr> gesetzt ist.
Im Fehlerfall = ""
- `jsonAsArrayDeleteObjectByIndex(<jsonstr>, <indexstr>)` : `jsonstring` //since 4.11.6.4: [W/L]
Liefert einen String mit dem in <jsonstr> übergebenen JSON Array bei dem das Objekt am Index <indexstr> entfernt wurde.
Im Fehlerfall = ""
- `jsonAsArrayToStringList(<jsonstr>)` : `stringlist` //since 4.11.6: [W/L]
Liefert einen String mit dem in <jsonstr> übergebenen JSON Array als Stringliste mit einem Arrayelement pro Zeile.
- `jsonStringListToJsonArray(<strlist>)` : `jsonstr` //since 4.11.6: [W/L]
Liefert einen String mit einem JSON Array der die Zeilen der Stringliste <strlist> als Elemente enthält.
- `jsonAsObjectGetKeyList(<jsonstr>)` : `stringlist` //since 4.11.6: [W/L]
Liefert eine Stringliste mit den Keys der im JSON Object <jsonstr> vorhandenen Keys.

Beispiel:

Lese productOnClients Objekte aus einer Datei, ändere die `clientId` auf den Wert der aktuellen Maschine und schreibe die Objekte über den opsi-webservice zurück.

```

DefVar $poc_file$
DefVar $objectStr$
DefVar $ArrayStr$
DefVar $pid$

DefStringlist $resultlist$
DefStringlist $resultlist1$
DefStringlist $productIdList$
DefStringlist $pocList$

Message "Delete productOnClient from opsi backend ..."
set $resultlist$ = getReturnListFromSection("opiservicecall_getPOC")
Set $ArrayStr$ = takestring(0, $resultlist$)
if not(jsonIsValid($ArrayStr$))
    LogError "got no valid json from Service"
    isFatalError
endif
if not(jsonisArray($ArrayStr$))
    LogError "got no json Array from Service"
    isFatalError
endif
comment "extract productIds ..."
comment "clean target list"

```

```

set $productIdList$ = emptylist($productIdList$)
comment "get stringlist "
set $pocList$ = jsonToArrayToStringList($ArrayStr$)
for %aktpoc% in $pocList$ do sub_fill_product_ids
for %aktProductId% in $productIdList$ do opsiServiceCall_del_productOnClient

Message "Restore productOnClient from file ..."
comment " get Restore data from file ..."
Set $ArrayStr$ = strLoadTextFile($poc_file$)
if not(jsonIsValid($ArrayStr$))
    LogError "got no valid json from file"
    isFatalError
endif
if not(jsonisArray($ArrayStr$))
    LogError "got no json Array from file"
    isFatalError
endif

comment "get list from array"
set $pocList$ = jsonToArrayToStringList($ArrayStr$)
comment "loop over list"
for %pocindex% = "0" to calculate(count($pocList$)+"-1") do sub_set_clientid_in_poclist
comment "convert modified list to json array"
set $ArrayStr$ = jsonStringListToJsonArray($pocList$)
set $ArrayStr$ = unquote2($ArrayStr$, "[]")
comment "write back"
opsiServiceCall_updatePOC

[sub_fill_product_ids]
set $objectstr$ = '%aktpoc%'
set $pid$ = jsonAsObjectGetValueByKey($objectstr$, "productId" )
set $productIdList$ = addToList($productIdList$, $pid$)

[sub_set_clientid_in_poclist]
set $objectstr$ = takeString("%pocindex%", $poclist$)
set $objectstr$ = jsonAsObjectSetStringtypeValueByKey(($objectstr$, "clientId", "%opsiserviceUser%"
)
set $poclist$ = setStringInListAtIndex($objectstr$, $poclist$, "%pocindex%")

[opsiServiceCall_updatePOC]
"method": "productOnClient_updateObjects"
"params": [
                                '$ArrayStr$',
                                ]

[opsiservicecall_getPOC]
"method": "productOnClient_getObjects"
"params": [
    "[]",
    '{"clientId": "%opsiserviceUser%", "productType": "LocalbootProduct"}'
]

[opsiServiceCall_del_productOnClient]
"method": "productOnClient_delete"
"params": [
                                '%aktProductId%',
                                '%opsiserviceuser%',

```

J

9.6 Umgang mit Zahlen [W/L]

Es gibt im *opsi-winst* keine speziellen Variablen für Zahlen. Es gibt allerdings einige Funktionen welche beim Umgang mit Zahlen helfen.

- `calculate(<str>)` [W/L]
Stringfunktion welche den arithmetischen Ausdruck im String `<str>` berechnet und als gerundeten integer string zurückgibt.
Intern werden die Berechnungen mit reellen Zahlen durchgeführt. Die Funktion kennt derzeit die Operatoren `+`, `-`, `*`, `/` sowie Klammern `(,)`.
Im Fehlerfall wird ein leerer String zurückgegeben und der Errorcounter um eins erhöht Enthält der übergebene String Zeichen die keine Zahlen oder gültige Operatoren sind, so ist dies ein Fehler.
Fehlt der zweite Operator, so wird hierfür der erste verwendet: `5+ = 10` ; `5* = 25`. Daher sollte beim Zusammensetzen des übergebenen strings verwendete Variablen z.B. mit der Funktion `isNumber` auf Gültigkeit geprüft werden.
Seit 4.11.3.5

Beispiele:

```
set $ConstTest$ = "0"
set $CompValue$ = calculate("-1+1")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $ConstTest$ = "1"
set $CompValue$ = calculate("0+1")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $ConstTest$ = "-1"
set $CompValue$ = calculate("0-1")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "5"
set $ConstTest$ = "25"
set $CompValue$ = calculate($string1$+"*"+$string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

```

set $string1$ = "5"
set $string2$ = "5"
set $ConstTest$ = "1"
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "5"
set $string2$ = "0"
set $ConstTest$ = ""
comment " expecting devision by zero error and empty string result"
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "9"
set $string2$ = "10"
set $ConstTest$ = "1"
comment "result 0.9 is rounded to 1 "
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "10"
set $string2$ = "9"
set $ConstTest$ = "1"
comment "result 1.1111 is rounded to 1 "
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "5"
set $string2$ = "5"
set $ConstTest$ = "55"
comment " rule * before +"
set $CompValue$ = calculate($string1$+"+"+$string2$+"*10")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

set $string1$ = "5"
set $string2$ = "5"
set $ConstTest$ = "100"

```

```

comment "brackets before rule * before + "
set $CompValue$ = calculate("("+$string1$+"*"+$string2$+")*10")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "ten"
set $ConstTest$ = ""
comment "invalid char error"
set $CompValue$ = calculate($string1$+"*"+$string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = ""
set $ConstTest$ = "25"
comment "5* is interpreted as 5*5"
set $CompValue$ = calculate($string1$+"*")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = ""
set $ConstTest$ = "10"
comment "5+ is interpreted as 5+5"
set $CompValue$ = calculate($string1$+"+")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $string1$ = "nothing"
set $string2$ = "foo"
set $ConstTest$ = ""
comment "invalid char error"
set $CompValue$ = calculate($string1$+"*"+$string2$)
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "foo"
set $ConstTest$ = ""
comment "invalid char error"
set $CompValue$ = calculate($string1$+"/"+$string2$)

```

```

if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

```

Für weitere Beispiele beachten Sie das Produkt *opsi-script-test* und dort den Bereich `$Flag_calculate$ = "on"`

Es gibt einen Vergleichsausdruck um zwei Strings wie (integer) Zahlen zu vergleichen. Wenn einer der Werte nicht in eine Zahl übertragen werden kann, wird ein Fehler ausgegeben.

Diese Zahlenvergleichsausdrücke haben die gleich Form wie die String-Vergleichsausdrücke, allerdings wird dem dem Vergleichszeichen ein INT vorangestellt:

<STRINGAUSDRUCK> INT<Vergleichszeichen> <STRINGAUSDRUCK>

So können Ausdrücke wie

```
if $Name1$ INT<= $Name2$
```

oder

```
if $Number1$ INT>= $Number2$
```

gebildet werden.

- `isNumber(<str>)` [W/L]
Boolsche Funktion welche *true* liefert wenn <str> einen ganzzahligen Wert (integer) repräsentiert.
Seit 4.11.3

siehe auch: Abschnitt [9.14.2](#)

9.7 Process und Script Functionen [W/L]

`waitForPackageLock(<seconds timeout string>,<bool should we kill>)` : `bool` //since 4.11.6.1 [L]

Liefert *true* zurück wenn das Linux Packagesystem nicht gesperrt ist. Ist es gesperrt, so wird <seconds timeout string> Sekunden auf die Freigabe gewartet. Ist der Timeout erreicht, so wird der Prozess welcher den Lock erzeugt hat abgeschossen wenn <bool should we kill> gleich *true* ist. Dieses Vorgehen ist aber nicht empfohlen.

`processIsRunning(<process name>)` : `boolean` //since 4.11.6.1 [W/L]

Liefert *true* wenn der Prozess <process name> in der aktuellen Prozessliste ist.

- `shellCall (<command string>)` : `stringlist (output)` //since 4.11.4.2 [W/L]
Führt <command string> mit der standard shell (cmd.exe / bash) aus

```
set $list$= shellCall('net start')
```

Ist eine Abkürzung für den Ausdruck:

```
set $list$ = getOutputStreamFromSection ('DosInAnIcon_netstart winst /sysnative')
```

```
[DosInAnIcon_netstart]
net start
```

- `shellCall (<command string>)` : `noresult` //since 4.11.6.1 [W/L]

```
shellCall('net start')
```

Ist eine Abkürzung für den Ausdruck:

```
DosInAnIcon_netstart winst /sysnative
```

```
[DosInAnIcon_netstart]
net start
```

- `shellCall (<command string>) : string (exitcode) //since 4.11.6.1 [W/L]`

```
set $exitcode$ = shellCall('net start')
```

Ist eine Abkürzung für den Ausdruck:

```
DosInAnIcon_netstart winst /sysnative
```

```
set $exitcode$ = getLastExitcode
```

```
[DosInAnIcon_netstart]
net start
```

- `powershellCall (<commandstr> [,<access str>=sysnative [,<policy bool str>=true]]) : stringlist (output) //since 4.12.0.16 [W]`
Führt <command string> mit der powershell aus.
Genauer gesagt wird ein Script ausgeführt das wie folgt aussieht:

```
trap { write-output $_ ; exit 1 }
<commandstr>
exit $LASTEXITCODE
```

Die erste Zeile dient dazu, das bei Exceptions kein `exitcode=0` zurückgeliefert wird und die letzte Zeile dient dazu den letzten produzierten exit code zurückzuliefern.

Die Architektur der powershell.exe ist per default `sysnative`. Die Architektur kann aber auch über den optionalen zweiten Parameter `<access str>` angegeben werden. Dabei muss der übergebene `<access str>` einer der folgenden Werte sein: `32bit`, `sysnative`, `64bit`.

(siehe auch: [Kapitel 64 Bit-Unterstützung](#))

Da per Windows Default die Execution Policy der Powershell auf `Restricted` steht können Scripte nicht ohne weiteres ausgeführt werden. Daher hat der Befehl `powershellCall` per default folgendes Verhalten: Die aktuelle execution-policy wird gesichert und die execution-policy auf `RemoteSigned` gesetzt. Dann wird das Script ausgeführt und anschließend die Execution-policy wieder zurück gesetzt. Dieses Defaultverhalten kann über den optionalen dritten Parameter `<policy bool str>` geändert werden: Hat `<policy bool str>` den Wert "false" so wird die execution-policy nicht modifiziert.

Wird die Funktion `powershellCall` dort aufgerufen wo eine Stringliste erwartet wird, so enthält die Stringliste die Ausgabe von `<commandstr>`.

Beispiel:

```
set $list$= powershellCall_list('Get-Process -ProcessName "opsi*")
```

Ist eine Abkürzung für den Ausdruck:

```
set $policy$ = takeString(0,shellCall('powershell.exe get-executionpolicy'))
shellCall('powershell.exe set-executionpolicy RemoteSigned')
set $list$ = getOutputStreamFromSection ('Execwith_ps powershell.exe winst /sysnative')
shellCall('powershell.exe set-executionpolicy '+$policy$)
```

[Execwith_ps]

```
trap { write-output $_ ; exit 1 }
Get-Process -ProcessName "opsi*"
exit $LASTEXITCODE
```

- `powershellCall (<commandstr> [,<access str>=sysnative [,<policy bool str>=true]])` : `noresult` //since 4.12.0.16 [W]
siehe [\[powershellCall_list\]](#)
Die Funktion `powershellCall` kann auch dort aufgerufen werden, wo kein Rückgabe wert erwartet wird.

Beispiel:

```
powershellCall('Get-Process -ProcessName "opsi*"')
```

- `powershellCall (<commandstr> [,<access str>=sysnative [,<policy bool str>=true]])` : `string (exitcode)` //since 4.12.0.16 [W]
siehe [\[powershellCall_list\]](#)
Wird die Funktion `powershellCall` dort aufgerufen wo eine String erwartet wird, so enthält der String den Exitcode des ausgeführten Scriptes.

Beispiel

```
set $exitcode$ = powershellCall('Get-Process -ProcessName "opsi*"')
```

- `processCall(<string>)` : `string (exitcode)` //since 4.11.6.1 [W/L]
Startet das Programm `<string>` als Prozess und liefert den Exitcode zurück.

```
set $exitcode$ = processCall('setup.exe /S')
```

Ist eine Abkürzung für den Ausdruck:

```
Winbatch_setup
set $exitcode$ = getLastExitcode
```

[Winbatch_setup]

```
setup.exe /S
```

Tatsächlich basiert `processCall` intern auf den selben Routinen wie `winbatch` und verarbeitet daher auch die selben Modifier:

- `/LetThemGo`
Verschiebt den aufgerufenen Prozess in den Hintergrund und wartet **nicht** auf dessen Beendigung; d.h. das sofort die nächste Zeile der WinBatch-Sektion bzw. die nächste Zeile des übergeordneten Programms abgearbeitet werden.
- `/TimeOutSeconds <seconds>`
Bricht das Warten auf das Prozessende oder eine Wartebedingung (`/WaitForProcessEnding`) nach Ablauf von `<seconds>` ab, auch wenn das Prozessende oder die Wartebedingung noch nicht erfüllt ist.
Der Prozess auf dessen Ende gewartet werden sollte wird nicht gestopt.
Kann seit Version 4.11.3 auch alleine (z.B. ohne `/WaitForProcessEnding`) verwendet werden, aber nicht zusammen mit `/WaitSeconds`.
Seit 4.11.4.6 wird der Zeitablauf bis zum Timeout über den Fortschrittsbalken angegeben.

- `/WaitSeconds` [number of seconds]
Die Parametrisierung `/WaitSeconds` [AnzahlSekunden] modifiziert das Verhalten dahingehend, dass *opsi-winst/opsi-script* jeweils erst nach [AnzahlSekunden] die Skriptbearbeitung fortsetzt. Die angegebene Zeit stoppt *opsi-winst/opsi-script* auf jeden Fall. In der Default-Einstellung wird zusätzlich auf das Ende der angestoßenen Prozesse gewartet. Ist letzteres nicht gewünscht, so kann der Parameter mit dem Parameter `/LetThemGo` kombiniert werden.
- `/WaitForProcessEnding` <program name>
Wartet darauf, das sich der Prozess mit dem Namen <program name> beendet.
Kann und sollte mit `/TimeOutSeconds` kombiniert werden.
- `/32Bit` //seit 4.11.3.5 [W]
Das ist der Default. Die in der Sektion angegebene Pfade werden als 32 Bit Pfade interpretiert.
Beispiel: `c:\windows\system32\regedit.exe` ruft (auch auf einem 64bit System) die 32 Bit *regedit.exe* auf.
- `/64Bit` //seit 4.11.3.5 [W]
Die in der Sektion angegebene Pfade werden als 64 Bit Pfade interpretiert.
Beispiel: `c:\windows\system32\regedit.exe` ruft (auf einem 64bit System) die 64 Bit *regedit.exe* auf.
- `/SysNative` //seit 4.11.3.5 [W]
Die in der Sektion angegebene Pfade werden gemäß der OS Architektur interpretiert.
Beispiel: `c:\windows\system32\regedit.exe` ruft auf einem 64bit System die 64 Bit *regedit.exe* und auf einem 32bit System die 32 Bit *regedit.exe* auf.

9.8 Spezielle Kommandos [W/L]

- `Killtask` <process> [W/L]
stoppt alle Prozesse, in denen das durch <process> bezeichnete Programm ausgeführt wird. Beispiel :

```
killtask "winword.exe"
```

- `ChangeDirectory` <directory> //since 4.11.2.6 [W/L]
Setzt das angegebene Directory als Arbeitsverzeichnis des *opsi-winst/opsi-script*. Wirkt auf alle nachfolgenden Aktionen (z.B. winbatch Sektionen) und wird am Ende eineses Scriptes automatisch zurückgesetzt. Beispiel :

```
ChangeDirectory "%SCRIPTPATH%\programm"
```

- `UpdateEnvironment` //since 4.11.5.1 [W/L]
Sendet Windows das Signal das Environment aus der Registry neu einzulesen. Anzuwenden nachdem Umgebungsvariablen wie z.B. PATH verändert, gesetzt oder gelöscht wurden. ABER: Normale DosBatch oder Winbatch Aufrufe erben trotzdem das *alte* Environment. Daher danach winbatch mit dem Parameter `/RunElevated` verwenden.

Example:

```
comment "Set Environment Variables and check for it ...."
Registry_add_environment /sysnative
UpdateEnvironment

comment "This will not work because the environment is inherited from the running process"
set $list$ = shellCall('set opsi-script-test')

comment "This will work because this new started process will get a new environment"
winbatch_check_environment /RunElevated
if ("42" = getlastExitCode)
    comment "passed"
```

```

else
    comment "failed"
endif

[Registry_add_environment]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment]
set "opsi-script-test"="deleteme"

[winbatch_check_environment]
"%system%\cmd.exe" /c "if %opsi-script-test%==deleteme exit 42"

```

- **sleepSeconds** <string> [W/L]
unterbricht die Programmausführung für <string> Sekunden. Dabei muß <string> ein Stringausdruck sein welcher eine Integerzahl repräsentiert.
- **markTime** [W/L]
Setzt einen Zeitstempel für die Systemlaufzeit und zeichnet diese auf.
- **diffTime** [W/L]
Zeichnet die vergangene Zeit seit der letzten aufgezeichneten Zeit (**marktime**) auf.

9.9 Kommandos zur Steuerung des Logging [W/L]

- **comment** <STRINGAUSDRUCK> [W/L]
bzw.
comment = <Zeichensequenz>
wird einfach der Wert des String-Ausdrucks (bzw. Zeichensequenz) als Kommentar in die Protokolldatei eingefügt.
- **LogError** <STRINGAUSDRUCK> [W/L]
oder
LogError = <Zeichensequenz>
Fügt eine zusätzlich Fehlermeldungen in der Protokolldatei ein und erhöht den Fehlerzähler um eins.
- **LogWarning** <STRINGAUSDRUCK> [W/L]
oder
LogWarning = <Zeichensequenz>
Fügt eine zusätzlich Warnmeldungen in der Protokolldatei ein und erhöht den Warnungszähler um eins.
- **includeLog** <file name> <tail size> //since 4.11.2.1 [W/L]
- **includeLog** <file name> <tail size> [<encoding>] //since 4.11.4.1 [W/L]
Fügt die Datei <file name> in den aktuellen ein. Dabei werden nur die letzten <tail size> Zeilen und nicht die komplette Logdatei eingefügt. Wenn Sie ein anderes Programm (z.B. ein setup programm) starten das eine Logdatei produziert, können Sie mit diesem Befehl die Informationen aus dieser Logdatei in den Log des *opsi-winst/opsi-script* übernehmen.
Seit Version 4.11.3.2 kann auch eine negative <tail size> angegeben werden. Dann arbeitet **includeLog** im *Head* Modus, d.h. ist <tail size> = -5, so werden die ersten 5 Zeilen von <file name> in den Log übernommen. Seit Version 4.11.4.1 kann als dritter Parameter ein encoding angegeben werden. Bekannte encodings sind beim Befehl **encoding** angegeben. Wird als encoding **auto** angegeben, so wird versucht das passende Encoding zu erraten. Beispiel:

```
includelog "%Scriptpath%\test-files\10lines.txt" "5"
includelog "%Scriptpath%\test-files\10lines_utf16.txt" "5" "ucs2be"
```

siehe: [Encoding](#)

- **SetConfidential** <secret string> [W/L]
Dient dazu vertrauliche Informationen (z.B. Passwörter) aus den Logdateien fernzuhalten. Diese werden dort durch (*confidential*) ersetzt.
Wir der Loglevel auf 9 gesetzt, so werden die 'confidential's im Klartext geloggt.
Seit Version 4.11.3.5

Beispiel:

```
message "SetConfidential"
SetConfidential "forbidden"
comment "This is a forbidden string"
comment "should be in the log file: This is a **(confidential)** string"
```

Log:

```
message SetConfidential
comment: This is a **(secret)** string
comment: should be in the log file: This is a **(confidential)** string
```

- **asConfidential**(<secret string expression>) : string //since 4.12.0.16 [W/L]
Diese Funktion dient dazu vertrauliche Informationen von einer Funktion zu erlangen, ohne das die Werte in der Logdatei landen. Dabei kommt es zu folgenden Ablauf:

1. Sichern des aktuellen Loglevels
2. Setzen des Loglevel auf Warning (4)
3. Auswerten der übergeben Stringexpression (z.B. einer Funktion welche einen String liefert)
4. Setzen dieses ermittelten Strings als Confidential d.h. dieser String wird ab jetzt nicht mehr geloggt.
5. Wiederherstellen des ursprünglichen Loglevels
6. Rückgabe des ermittelten Stringwertes.

Beispiel:

```
set $ConstTest$ = asConfidential(stringReplace("this is my old secret", "old", "new"))
comment "this is my new secret"
comment "should be in the log file: **(confidential)** "
```

Log:

```
Set $ConstTest$ = asConfidential(stringReplace("this is my old secret", "old", "new"))
The value of the variable "$ConstTest$" is now: **(confidential)**
comment: This is a **(confidential)** string
comment: should be in the log file: This is a **(confidential)** string----
```

see also : [\[SetConfidential\]](#) see also : [\[GetConfidentialProductProperty\]](#)

9.10 Anweisungen für Information und Interaktion [W/L]

- **Message** <STRINGAUSDRUCK> [W/L]

bzw.

Message = <Buchstabenfolge>

bewirkt, dass in der Batch-Oberfläche des *opsi-winst/opsi-script* der Wert von STRINGAUSDRUCK bzw. dass die Buchstabenfolge als Hinweis-Zeile zum gerade laufenden Installationsvorgang angezeigt wird (solange bis die Installation beendet ist oder eine andere **message** angefordert wird).

Empfohlen ist die erste Variante, da nur hier auch Variablen für den Messagetext verwendet werden können. Beispiel:

```
Message "Installation von "+$productid$
```

- **ShowMessageFile** <file name> [W/L]

zeigt den Inhalt der Datei <file name> in einem gesonderten Fenster an und wartet auf Bestätigung durch den Anwender. Z.B. könnte so eine "Nachricht des Tages" angezeigt werden:

```
ShowMessageFile "p:\login\day.msg"
```

- **ShowBitmap** [<DATEINAME>] [<Beschriftung>] [W/L]

wird die Bilddatei <DATEINAME> (BMP-, JPEG- oder PNG-Format, 160x160 Pixel) in der Batch-Oberfläche angezeigt. Eine Beschriftung kann ebenfalls hinzugefügt werden. <DATEINAME> und <Beschriftung> sind String-Ausdrücke. Wenn der Namensparameter fehlt, wird das Bild gelöscht. Beispiel:

```
ShowBitmap "%scriptpath%" + $ProduktName$ + ".bmp" "$ProduktName$"
```

- **Pause** <STRINGAUSDRUCK> [W/L]

bzw.

Pause = <Zeichensequenz>

Die beiden Anweisungen zeigen in einem Textfenster den in STRINGAUSDRUCK gegebenen Text (bzw. Zeichensequenz) an. Auf Anklicken eines Knopfes setzt sich der Programmablauf fort.

- **Stop** <STRINGAUSDRUCK> [W/L]

bzw.

stop = <Zeichensequenz> [W/L]

der STRINGAUSDRUCK (bzw. Zeichensequenz, die möglicherweise auch leer ist) wird angezeigt und um Bestätigung gebeten, dass der Programmablauf abgebrochen werden soll.

- **setActionProgress** <string> : noresult //since 4.11.3 [W/L]

Übermittelt <string> als ActionProgress für das laufende Produkt an den Server. Der Default von ActionProgress ist *installing* während ein Script ausgeführt wird. Der ActionProgress wird im configed angezeigt.

9.11 Commands for userLoginScripts / User Profile Management [W]

- **GetScriptMode** //since 4.11.2.1

liefert eines der beiden Werte *Machine,Login*:

– *Machine* - das Script läuft **nicht** als *userLoginScript*

– *Login* - das Script läuft als *userLoginScript*

- `GetUserSID(<Windows Username>)`
siehe auch [\[GetUserSID\]](#)

- `GetLoggedInUser //since 4.11.1.2`

- `GetUsercontext //since 4.11.1.2`
liefert den Namen des Users in dessen Kontext der *opsi-winst/opsi-script* gerade läuft.
siehe auch: [\[GetUsercontext\]](#)

- `saveVersionToProfile //since 4.11.2.1`
speichert `productversion-packageversion` die im lokalen Profil
Diese Funktion ist gedacht für *userLoginScripts*.
Diese Funktion kann in Kombination mit `readVersionFromProfile` verwendet werden um festzustellen ob ein Script schonmal gelaufen ist. Es speichert im Lokalen Profil (in der Datei "%CurrentAppdata-Dir%\opsi.org\userLoginScripts.ini"), dass das *userLoginScript* für dieses opsi product in dieser product version und package version für den aktuellen user ausgeführt wurde. Sieh auch `scriptWasExecutedBefore`

- `readVersionFromProfile //since 4.11.2.1`
liefert einen string mit `productversion-packageversion` für das aktuelle opsi produkt der aus dem lokalen Profil ausgelesen wird. Siehe auch: `saveVersionToProfile`
Diese Funktion ist gedacht für *userLoginScripts*.

- `scriptWasExecutedBefore //since 4.11.2.1`
Mit dieser boolschen Funktion kann überprüft werden, ob das *userLoginScript* zu diesem Produkt in dieser Version schon mal zuvor gelaufen ist und eine erneute Ausführung unnötig ist. Dazu liest diese Funktion zunächst einen evtl. vorhandenen Versionsstempel vom Profil ein (wie das mit `readVersionFromProfile` möglich ist) und vergleicht diesen mit der aktuell laufenden Version. Aus dem Vergleich ergibt sich der Rückgabewert (wahr/falsch). Danach werden noch die aktuellen Werte in das Profil zurückgeschrieben (wie das mit `saveVersionToProfile` möglich ist). Somit benötigen Sie nur diese eine Funktion in einer `if` Anweisung, um zu prüfen ob das Script schon mal gelaufen ist.

- `isLoginScript //since 4.11.2.1`
Diese boolsche Funktion liefert `true` wenn das aktuelle Script als *userLoginScript* läuft. Siehe auch: `GetScriptMode`

9.12 For-To Schleife [W/L]

Zum mehrfachen Ausführen eines Befehls oder einer Subsektion.

Syntax:

```
for %<temporary string variable>% = <start string> to <end string> do <one statement> //since 4.11.5 [W/L]
```

Die temporäre Variable `%<temporary string variable>%` muss nicht und darf nicht deklariert werden und ist in der aufgerufenen Subsektion als Konstante verfügbar.

Example:

Code from opsi-script-test:

```

message "for to loop"
set $ConstTest$ = "12345"
set $CompValue$ = ""
for %s% = "1" to "5" do sub_iteration_test
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

[sub_iteration_test]
set $CompValue$ = $CompValue$ + '%s%'

```

ergibt den Log:

```

message for to loop
Set $ConstTest$ = "12345"
  The value of the variable "$ConstTest$" is now: "12345"
Set $CompValue$ = ""
  The value of the variable "$CompValue$" is now: ""

~~~~~ Looping through: '1', '2', '3', '4', '5'

~~~~~ Start Sub ~~~~~ sub_iteration_test
Set $CompValue$ = $CompValue$ + '1'
  The value of the variable "$CompValue$" is now: "1"

~~~~~ End Sub ~~~~~ sub_iteration_test

~~~~~ Start Sub ~~~~~ sub_iteration_test
Set $CompValue$ = $CompValue$ + '2'
  The value of the variable "$CompValue$" is now: "12"

~~~~~ End Sub ~~~~~ sub_iteration_test

~~~~~ Start Sub ~~~~~ sub_iteration_test
Set $CompValue$ = $CompValue$ + '3'
  The value of the variable "$CompValue$" is now: "123"

~~~~~ End Sub ~~~~~ sub_iteration_test

~~~~~ Start Sub ~~~~~ sub_iteration_test
Set $CompValue$ = $CompValue$ + '4'
  The value of the variable "$CompValue$" is now: "1234"

~~~~~ End Sub ~~~~~ sub_iteration_test

~~~~~ Start Sub ~~~~~ sub_iteration_test
Set $CompValue$ = $CompValue$ + '5'
  The value of the variable "$CompValue$" is now: "12345"

~~~~~ End Sub ~~~~~ sub_iteration_test

```

```

~~~~~ End Loop
If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: passed
Else
EndIf

```

9.13 Switch / Case Statement [W/L]

Syntax:

```

Switch <string expression>
Case <string const>
<statement(s)>
EndCase
[DefaultCase
<statement(s)>
EndCase ]
EndSwitch

```

Examples:

Code from opsi-script-test:

```

set $ConstTest$ = "5"
Switch $ConstTest$
  Case "1"
    set $CompValue$ = "1"
  EndCase
  Case "2"
    set $CompValue$ = "2"
  EndCase
  Case "3"
    set $CompValue$ = "3"
  EndCase
  Case "4"
    set $CompValue$ = "4"
  EndCase
  Case "5"
    set $CompValue$ = "5"
  EndCase
  Case "6"
    set $CompValue$ = "6"
  EndCase
  Case "7"
    set $CompValue$ = "7"
  EndCase
  DefaultCase
    set $CompValue$ = "notexisting"
  EndCase
EndSwitch
if ($ConstTest$ = $CompValue$)

```

```

        comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

```

```

[Sub_check_exitcode]
comment "Test for installation success via exit code"
set $ExitCode$ = getLastExitCode
; informations to exit codes see
; http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx
; http://msdn.microsoft.com/en-us/library/aa368542.aspx
Switch $ExitCode$
    Case "0"
        comment "Looks good: setup program gives exitcode zero"
    EndCase
    Case "1605"
        comment "ERROR_UNKNOWN_PRODUCT 1605"
        comment "This action is only valid for products that are currently installed."
        comment "Uninstall of a not installed product failed - no problem"
    EndCase
    Case "1641"
        comment "looks good: setup program gives exitcode 1641"
        comment "ERROR_SUCCESS_REBOOT_INITIATED 164"
        comment "The installer has initiated a restart."
        comment "This message is indicative of a success."
        ExitWindows /Reboot
    EndCase
    Case "3010"
        comment "looks good: setup program gives exitcode 3010"
        comment "ERROR_SUCCESS_REBOOT_REQUIRED 3010"
        comment "A restart is required to complete the install."
        comment "This message is indicative of a success."
        ExitWindows /Reboot
    EndCase
    DefaultCase
        logError "Fatal: Setup program gives an unknown exitcode unequal zero: " + $
ExitCode$
        isFatalError "Exit Code: " + $ExitCode$
    EndCase
EndSwitch

```

9.14 Bedingungsanweisungen (if-Anweisungen) [W/L]

Die Ausführung einer oder mehrere Anweisungen kann in den primären Sektionen von der Erfüllung bzw. Nichterfüllung einer Bedingung abhängig gemacht werden.

Beispiel:

```

;Welche Windows-Version?
DefVar $MSVersion$

Set $MSVersion$ = GetMsVersionInfo

```

```

if CompareDotSeparatedNumbers($MSVersion$, ">=", "6")
    sub_install_win7
else
    if ( $MSVersion$ = "5.1" )
        sub_install_winXP
    else
        stop "not a supported OS-Version"
    endif
endif
endif

```

9.14.1 Allgemeine Syntaxregeln [W/L]

Folgendes Schema der if-Anweisung ist ersichtlich:

```

if <Bedingung>
;eine oder mehrere Anweisungszeilen
else
;eine oder mehrere Anweisungszeilen
endif

```

Der `else`-Teil der Anweisung darf fehlen.

if-Anweisungen können geschachtelt werden. Das bedeutet, dass in der Anweisung nach einem `if` Satz (sowohl in dem `if` als auch im `else` Teil) eine weitere if-Anweisung folgen darf.

<Bedingungen> sind boolesche Ausdrücke, das heißt logische Ausdrücke, die entweder den Wert *wahr* oder den Wert *falsch* tragen können.

9.14.2 Boolesche Ausdrücke [W/L]

Ein Vergleichsausdruck, welcher ein boolescher Ausdruck ist, sieht folgendermaßen aus:

<STRINGAUSDRUCK> <Vergleichszeichen> <STRINGAUSDRUCK>

An der Stelle <Vergleichszeichen> kann eins der folgenden Zeichen stehen:

< <= >= >

Bei String-Vergleichen im *opsi-winst/opsi-script* wird Groß-/Kleinschreibung nicht unterschieden.

Ungleich muss mit einem `NOT()` Ausdruck umgesetzt werden, was weiter unten gezeigt wird.

Es gibt einen Vergleichsausdruck um zwei Strings wie (integer) Zahlen zu vergleichen. Wenn einer der Werte nicht in eine Zahl übertragen werden kann, wird ein Fehler ausgegeben.

Diese Zahlenvergleichsausdrücke haben die gleich Form wie die String-Vergleichsausdrücke, allerdings wird dem dem Vergleichszeichen ein `INT` vorangestellt:

<STRINGAUSDRUCK> INT<Vergleichszeichen> <STRINGAUSDRUCK>

So können Ausdrücke wie

```
if $Name1$ INT<= $Name2$
```

oder

```
if $Number1$ INT>= $Number2$
```

gebildet werden.

Boolesche Operator sind `AND`, `OR` und `NOT()` (Groß-/Kleinschreibung nicht unterschieden).

`b1`, `b2` und `b3` sind boolesche Ausdrücke, die sich zu kombinierten Ausdrücken verbinden lassen.

`b1 AND b2`

`b1 OR b2`

`NOT(b3)`

Diese booleschen Ausdrücke zeigen dabei eine Konjunktion (`AND`), ein Disjunktion (`OR`) und eine Negation (`NOT`).

Ein boolescher Ausdruck kann in runden Klammer eingeschlossen werden (diese produziert dann einen neuen booleschen Ausdruck mit dem selben Wert).

Die allgemeine Regel für boolesche Operatorprioritäten ("and" vor "or") sind im Moment nicht implementiert. Ein Ausdruck mit mehr als einem Operator wird von links nach rechts interpretiert. Wenn also ein boolescher Ausdruck einen AND und OR Operator enthalten soll, müssen runde Klammern eingesetzt werden. So muss zum Beispiel explizit geschrieben werden

```
b1 OR (b2 AND b3)
```

oder

```
(b1 OR b2) AND b3
```

Das zweite Beispiel beschreibt, was ausgeführt werden würde, wenn keine runden Klammern gesetzt wäre – wohingegen die übliche Operatorprioritäten so laufen würde wie in der ersten Zeile angezeigt.

Boolesche Operatoren können als spezielle boolesche Wertefunktionen eingesetzt werden (die Negation-Operatoren demonstrieren das sehr deutlich).

Es sind noch weitere boolesche Funktionen implementiert. Jeder Aufruf einer solchen Funktion begründet sich in einem booleschen Ausdruck:

- `FileExists(<datei name>)` [W/L]
Die Funktion gibt wahr zurück, wenn die genannte Datei oder das Verzeichnis existiert, ansonsten kommt die Antwort falsch.
- `FileExists32(<datei name>)` [W] siehe [Kapitel 64 Bit-Unterstützung](#)
- `FileExists64(<datei name>)` [W] siehe [Kapitel 64 Bit-Unterstützung](#)
- `FileExistsSysNative(<datei name>)` [W] siehe [Kapitel 64 Bit-Unterstützung](#)

- `LineExistsIn(<Zeile>, <Dateiname>)` [W/L]
Die Funktion gibt wahr zurück, wenn die Textdatei <Dateiname> eine Zeile beinhaltet, die im ersten Parameter beschrieben ist. Anderenfalls (oder falls die Datei garnicht existiert) wird falsch zurückgegeben.
Jeder Parameter ist ein String-Ausdruck.

- `LineBeginning_ExistsIn(<string>, <Dateiname>)` [W/L]
Die Funktion gibt wahr zurück, wenn in der Textdatei <Dateiname> eine Zeile vorhanden ist, welche mit dem Parameter <string> beginnt. Anderenfalls (oder falls die Datei garnicht existiert) wird falsch zurückgegeben.
Jeder Parameter ist ein String-Ausdruck.

- `LineContaining_ExistsIn(<string>, <Dateiname>)` //since 4.11.4.10 [W/L]
Die Funktion gibt wahr zurück, wenn in der Textdatei <Dateiname> eine Zeile vorhanden ist, welche den Parameter <string> enthält. Anderenfalls (oder falls die Datei garnicht existiert) wird falsch zurückgegeben.
Jeder Parameter ist ein String-Ausdruck.

- `XMLAddNamespace(<XMLfilename>, <XMLelementname>, <XMLnamespace>)` [W]
Mit dieser Funktion wird ein XML Namensraum im ersten XML-Element-Tag mit dem vergebenen Namen definiert (falls noch nicht vorhanden). Er wird ausgegeben, wenn ein Name eingefügt wurde. Die *opsi-winst/opsi-script* XML-Patch-Sektion benötigt diese Namensraumdefinition. Der File muss so formatiert werden, dass das Element-Tag keine Zeilenumbrüche beinhaltet. Für ein Anwendungsbeispiel siehe im Kochbuch [Kapitel "Einfügen einer Namensraumdefinition in eine XML-Datei"](#).

- `XMLRemoveNamespace(<XMLfilename>, <XMLelementname>, <XMLnamespace>)` [W]
Mit dieser Funktion wird die Definition des XML Namensraum wieder entfernt. Es wird wahr ausgegeben, wenn eine Entfernung erfolgt ist. (Beispiel im Kochbuch siehe [Kapitel "Einfügen einer Namensraumdefinition in eine XML-Datei"](#)).

- `HasMinimumSpace(<Laufwerksname>, <Kapazität>)` [W]
gibt `true` zurück, wenn mehr Platz als die geforderte `<Kapazität>` auf dem Laufwerk `<Laufwerksname>` vorhanden ist. `<Kapazität>` ist syntaktisch ebenso wie `<Laufwerksname>` ein String-Ausdruck. Die `<Kapazität>` kann als Nummer ohne genauere Bezeichnung (dann interpretiert als Bytes) oder mit einer näheren Bezeichnung wie "kB", "MB" oder "GB" ausgegeben werden (case sensitive).
Anwendungsbeispiel:

```
if not (HasMinimumSpace ("%SYSTEMDRIVE%", "500 MB"))
  LogError "Es ist nicht genug Platz auf dem Laufwerk %SYSTEMDRIVE%, erforderlich sind 500 MB"
  isFatalError
endif
```

- `opsiLicenseManagementEnabled` [W/L]
gibt `true` zurück wenn das opsi System über ein anwendbares (freigeschaltetes) Lizenzmanagement verfügt.

```
if opsiLicenseManagementEnabled
  set $mykey$ = DemandLicenseKey ("pool_office2007")
else
  set $mykey$ = GetProductProperty("productkey", "")
endif
```

- `runningAsAdmin` [W]
Boolsche Funktion welche `true` liefert wenn das laufende Script mit Administrativen Rechten ausgeführt wird.
Seit 4.11.1.1
- `isLoginScript` [W]
Boolsche Funktion welche `true` liefert wenn das laufende Script über die opsi Erweiterung *User Profile Management* als *userLoginScript* läuft.
Seit 4.11.2.1
siehe auch : [\[isLoginScript\]](#)
- `contains(<str>, <substr>)` [W/L]
Boolsche Funktion welche `true` liefert wenn `<substr>` in `<str>` enthalten ist. Die Funktion arbeitet case sensitive.
Seit 4.11.3
siehe auch : [\[contains\]](#)
- `isNumber(<str>)` [W/L]
Boolsche Funktion welche `true` liefert wenn `<str>` einen ganzzahligen Wert (integer) representiert.
Seit 4.11.3
siehe auch : [\[isNumber\]](#)
- `runningOnUefi` [W]+ Boolsche Funktion welche `true` liefert wenn das laufende Betriebssystem im UEFI mode gebootet wurde.
Seit 4.11.4.3
- `runningInPE` //since 4.12.0.13: [W/L]
Boolsche Funktion welche `true` liefert wenn das laufende Betriebssystem ein Windows PE ist.
- `isDriveReady(<drive letter>)` //since 4.11.4.4: [W]
Boolsche Funktion welche `true` liefert auf das angebene Laufwerk zugegriffen werden kann (z.B. In Wechsellaufwerk ist ein Medium ist eingelegt)

- `saveTextFile(<list>, <filename>)` //since 4.11.4.4: [W/L]
Boolsche Funktion welche *true* liefert wenn die Liste <list> erfolgreich in <filename> gespeichert wurde.
- `saveTextFileWithEncoding(<list>, <filename>, <encoding>)` : bool //since 4.11.6.4
Boolsche Funktion welche *true* liefert wenn die Liste <list> erfolgreich in dem dem Zeichensatz <encoding> nach <filename> gespeichert wurde.
- `CompareDotSeparatedNumbers(<str1>,<relation str>,<str2>)` //since 4.11.5.2: [W/L]
vergleicht zwei Strings vom Typ <zahl>[.<zahl>[.<zahl>[.<zahl>]]] unter Verwendung des <relation str> der eins von den folgenden Werten haben darf: [<, <=>, =, >=>, >].
siehe auch: Stringfunktion `CompareDotSeparatedNumbers(<string1>, <string2>)` siehe auch: Stringfunktion `CompareDotSeparatedStrings(<string1>, <string2>)` : [[CompareDotSeparatedStrings_str](#)] siehe auch : [[CompareDotSeparatedStrings_bool](#)] siehe auch : [[CompareDotSeparatedNumbers_str](#)]

Example:

The code:

```

set $string1$ = "1.2.30.4.5"
set $string2$ = "1.20.30.4.5"
if CompareDotSeparatedNumbers($string1$, "<", $string2$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
if CompareDotSeparatedNumbers($string1$, "<=", $string2$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
if CompareDotSeparatedNumbers($string1$, "=", $string2$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
if CompareDotSeparatedNumbers($string1$, ">=", $string2$)
    set $TestResult$ = "not o.k."
    LogWarning "failed"
else
    comment "passed"
endif
if CompareDotSeparatedNumbers($string1$, ">", $string2$)
    set $TestResult$ = "not o.k."
    LogWarning "failed"
else
    comment "passed"
endif

```

```

if CompareDotSeparatedNumbers($string1$, ">", $string2$)
    set $TestResult$ = "not o.k."
    LogWarning "failed"
else
    comment "passed"
endif

```

produce the log:

```

Set $string1$ = "1.2.30.4.5"
The value of the variable "$string1$" is now: "1.2.30.4.5"
Set $string2$ = "1.20.30.4.5"
The value of the variable "$string2$" is now: "1.20.30.4.5"
If
    Checking if "1.2.30.4.5" is "<" than / as "1.20.30.4.5"
    CompareDotSeparatedNumbers($string1$, "<", $string2$) <<< result true
Then
    comment: passed
Else
EndIf
If
    Checking if "1.2.30.4.5" is "<=" than / as "1.20.30.4.5"
    CompareDotSeparatedNumbers($string1$, "<=", $string2$) <<< result true
Then
    comment: passed
Else
EndIf
If
    Checking if "1.2.30.4.5" is "<=" than / as "1.20.30.4.5"
    CompareDotSeparatedNumbers($string1$, "<=", $string2$) <<< result true
Then
    comment: passed
Else
EndIf
If
    Checking if "1.2.30.4.5" is "=" than / as "1.20.30.4.5"
    CompareDotSeparatedNumbers($string1$, "=", $string2$) <<< result false
Then
Else
    comment: passed
EndIf
If
    Checking if "1.2.30.4.5" is ">=" than / as "1.20.30.4.5"
    CompareDotSeparatedNumbers($string1$, ">=", $string2$) <<< result false
Then
Else
    comment: passed
EndIf
If
    Checking if "1.2.30.4.5" is ">=" than / as "1.20.30.4.5"
    CompareDotSeparatedNumbers($string1$, ">=", $string2$) <<< result false
Then
Else
    comment: passed
EndIf
If
    Checking if "1.2.30.4.5" is ">" than / as "1.20.30.4.5"

```

```

CompareDotSeparatedNumbers($string1$, ">", $string2$) <<< result false
Then
Else
    comment: passed
EndIf

```

- `CompareDotSeparatedStrings(<str1>,<relation str>,<str2>)` //since 4.11.5.2: [W/L]
vergleicht zwei Strings vom Typ `<str>[.<str>[.<str>[.<str>]]]` unter Verwendung des `<relation str>` der eins von den folgenden Werten haben darf: [`<`,`←`,`=`,`→`,`>`].
siehe auch: Stringfunktion `CompareDotSeparatedStrings(<string1>, <string2>)` : [\[CompareDotSeparatedStrings_str\]](#) siehe auch : [\[CompareDotSeparatedNumbers_bool\]](#) siehe auch : [\[CompareDotSeparatedNumbers_str\]](#)
- `boolToString(<boolean expression>)` : bool string (true/false) // since 4.12.0.0 [W/L]
- `stringToBool(<string expression: true/false>)` : boolean // since 4.12.0.0 [W/L]
- `RegKeyExists(<regkey>[,<access str>])` : bool //since 4.12.0.16 [W]
Prüft ob der als `<regkey>` übergeben String als Registrykey existiert. Wird der Registrykey gefunden so wird `true` zurückgegeben, ansonsten `false`.
Die Zugriffsart ist per Default `sysnative`. Über den optionalen zweiten Parameter kann die Zugriffsart auch explizit angegeben werden. Dabei muss der übergebene `<access str>` einer der folgenden Werte sein: `32bit`, `sysnative`, `64bit`.
(siehe auch: [Kapitel 64 Bit-Unterstützung](#))

Beispiele:

```

RegKeyExists("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon")
RegKeyExists("HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general", "32bit")

```

- `RegVarExists(<regkey>, <var str> [,<access str>])` : bool //since 4.12.0.16 [W]
Prüft ob der als `<regkey>` übergeben String als Registrykey existiert und ob dort eine Variable mit dem Namen `<var str>` existiert. Wird beides gefunden so wird `true` zurückgegeben, ansonsten `false`.
Die Zugriffsart ist per Default `sysnative`. Über den optionalen dritten Parameter kann die Zugriffsart auch explizit angegeben werden. Dabei muss der übergebene `<access str>` einer der folgenden Werte sein: `32bit`, `sysnative`, `64bit`.
(siehe auch: [Kapitel 64 Bit-Unterstützung](#))

Beispiele:

```

RegVarExists("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon", "Shell")
RegVarExists("HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general", "bootmode", "32bit")

```

9.15 Include Kommandos [W/L]



Achtung

Die Verwendung von Include Kommandos führt schnell zu unübersichtlichen Code. Lassen Sie die Finger davon wenn Sie Anfänger sind.

9.15.1 Include Kommandos: Syntax

Mit Include Kommandos kann der Inhalt einer externen Datei dem laufende Script hinzugefügt werden. Dies kann entweder einfügend oder anhängend erfolgen. Die Include Kommandos sind normale Kommandos der primären Sektionen. Die eingefügten Dateien können weitere Include Kommandos enthalten.

Diese Kommandos gibt es seit Version 4.11.3

- `include_insert <file name> [W/L]`
Fügt den Inhalt von <file name> nach der aktuellen Zeile im laufenden Script ein. Somit ist die erste Zeile der eingefügten Datei die nächste Zeile welche der *opsi-winst/opsi-script* interpretiert.
- `include_append <file name> [W/L]`
Fügt den Inhalt von <file name> am Ende des laufenden Scriptes ein. Diese Anweisung dient vor allem dazu Sektionen aus z.B. einer Bibliothek hinzu zufügen.

Für beide Funktionen gilt:

<file name> ist:

- Ein kompletter Pfad zu einer Datei. [W/L]
- Eine Datei in `%ScriptPath%` [W/L]
- Eine Datei in `%opsiScriptHelperPath%\lib` [W]
Entspricht: `%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib`
- Eine Datei in `%ScriptPath%/. /lib //since 4.11.5.2` [W/L]
- Eine Datei in `%WinstDir%\lib` [W]

Die Prüfung erfolgt in dieser Reihenfolge. Die erste Datei die gefunden wird, wird genommen.

Beispiel:

Wir haben folgendes Script:

```
[Actions]
include_append "section_Files_del_tmp_dummy.opsiinc"
include_insert "include-test1.opsiinc"
```

Dabei hat die Datei `include-test1.opsiinc` folgenden Inhalt:

```
DefVar $inctestvar$
set $inctestvar$ = "inctest"
Files_del_tmp_dummy
include_append "section_Files_copy_inctest.opsiinc"
Files_copy_inctest

if fileExists("c:\opsi.org\tmp\dummy.txt")
    comment "passed"
else
    comment "failed"
    set $TestResult$ = "not o.k."
    LogWarning "include test failed"
endif

if fileExists("%scriptpath%\test-files\dummy.txt")
    comment "passed"
else
    comment "failed"
```

```

        set $TestResult$ = "not o.k."
        LogWarning "include test failed"
    endif
Files_del_tmp_dummy

```

Dabei hat die Datei section_Files_copy_inctest.opsiinc folgenden Inhalt:

```

[Files_copy_inctest]
copy "%scriptpath%\test-files\dummy.txt" "c:\opsi.org\tmp"

```

Dabei hat die Datei section_Files_del_tmp_dummy.opsiinc folgenden Inhalt:

```

[Files_del_tmp_dummy]
del -f "c:\opsi.org\tmp\dummyt.txt"

```

9.15.2 Include Kommandos: Library

Mit der Version 4.11.3 werden folgende Includefiles in %WinstDir%\lib ausgeliefert:

insert_check_exit_code.opsiinc:

```

; opsi include file

DefVar $ExitCode$

include_append "section_sub_check_exitcode.opsiinc"

```

insert_get_licensekey.opsiinc:

```

; opsi include file

DefVar $LicenseRequired$
DefVar $LicenseKey$
DefVar $LicensePool$

include_append "section_sub_get_licensekey.opsiinc"

```

section_sub_check_exit_code.opsiinc:

```

;opsi include file

[Sub_check_exitcode]
comment "Test for installation success via exit code"
set $ExitCode$ = getLastExitCode
; informations to exit codes see
; http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx
; http://msdn.microsoft.com/en-us/library/aa368542.aspx
if ($ExitCode$ = "0")
    comment "Looks good: setup program gives exitcode zero"
else
    comment "Setup program gives a exitcode unequal zero: " + $ExitCode$
    if ($ExitCode$ = "1605")
        comment "ERROR_UNKNOWN_PRODUCT 1605 This action is only valid for products
that are currently installed."
        comment "Uninstall of a not installed product failed - no problem"
    else
        if ($ExitCode$ = "1641")

```

```

        comment "looks good: setup program gives exitcode 1641"
        comment "ERROR_SUCCESS_REBOOT_INITIATED 1641 The installer has
initiated a restart. This message is indicative of a success."
        ExitWindows /Reboot
    else
        if ($ExitCode$ = "3010")
            comment "looks good: setup program gives exitcode 3010"
            comment "ERROR_SUCCESS_REBOOT_REQUIRED 3010 A restart is
required to complete the install. This message is indicative of a success."
            ExitWindows /Reboot
        else
            logError "Fatal: Setup program gives an unknown exitcode unequal
zero: " + $ExitCode$
            isFatalError "Exit Code: "+ $ExitCode$
        endif
    endif
endif
endif
endif

```

section_sub_get_licensekey.opsiinc:

```

; opsi include file

[Sub_get_licensekey]
if opsiLicenseManagementEnabled
    comment "License management is enabled and will be used"

    comment "Trying to get a license key"
    Set $LicenseKey$ = demandLicenseKey ($LicensePool$)
    ; If there is an assignment of exactly one licensepool to the product the following call
is possible:
    ; Set $LicenseKey$ = demandLicenseKey ("", $ProductId$)
    ;
    ; If there is an assignment of a license pool to a windows software id, it is possible to
use:
    ; DefVar $WindowsSoftwareId$
    ; $WindowsSoftwareId$ = "..."
    ; Set $LicenseKey$ = demandLicenseKey ("", "", $WindowsSoftwareId$)

    DefVar $ServiceErrorClass$
    set $ServiceErrorClass$ = getLastServiceErrorClass
    comment "Error class: " + $ServiceErrorClass$

    if $ServiceErrorClass$ = "None"
        comment "Everything fine, we got the license key '" + $LicenseKey$ + "'"
    else
        if $ServiceErrorClass$ = "LicenseConfigurationError"
            LogError "Fatal: license configuration must be corrected"
            LogError getLastServiceErrorMessage
            isFatalError $ServiceErrorClass$
        else
            if $ServiceErrorClass$ = "LicenseMissingError"
                LogError "Fatal: required license is not supplied"
                isFatalError $ServiceErrorClass$
            endif
        endif
    endif
endif
endif

```

```

else
    LogError "Fatal: license required, but license management not enabled"
    isFatalError "No Licensemanagement"
endif

```

9.16 Aufrufe von Unterprogrammen [W/L]

Anweisungen in primären Sektionen, die auf einen Programmtext an anderer Stelle verweisen, sollen hier Unterprogramm- oder Prozeduraufrufe heißen.

```

if ( $MSVersion$ >= "6" )
    sub_install_win7
else
    if ( $MSVersion$ = "5.1" )
        sub_install_winXP
    else
        stop "not a supported OS-Version"
    endif
endif
endif

```

So "ruft" in obigem Beispiel die Anweisung in der Actions-Sektion

```
sub_install_winXP
```

die Sektion `[sub_install_winXP]`, welche dann im Skript an anderer Stelle nachzulesen ist als

```

[sub_install_winXP]
Files_Kopieren_XP
WinBatch_SetupXP

```

Weil es sich in diesem Beispiel um eine Sub-Sektion handelt, also immer noch um eine primäre Sektion, kann in ihr wiederum auf weitere Sektionen verwiesen werden, in diesem Fall auf die Sektionen `[Files_Kopieren_XP]` und `[WinBatch_Setup_XP]`.

Generell gibt es drei Wege um die genannten Anweisungen zu platzieren:

1. Der gebräuchlichste Ort für den Aufruf einer Sub-Sektion ist eine weitere interne Sektion im Skript, wo die aufgerufene Befehle platziert werden (wie in dem Beispiel).
2. Die bezeichneten Befehle können auch in einer anderen Datei untergebracht werden, welche als externe Sektion läuft.
3. Jede String-Liste kann als eine Befehlsliste für einen Sub-Programm Aufruf benutzt werden.

Zur Syntax der Sub-Programm Aufrufe im einzelnen:

9.16.1 Komponenten eines Unterprogrammaufrufs

Formal kann die Syntax wie folgt aufgerufen werden
`<proc. type>(<proc. name> | <External proc. file> | <Stringlisten Funktion>)`

Diese Ausdrücke können durch einen oder mehrere Parameter ergänzt werden (ist vom Ablauftyp abhängig).

Das bedeutet: Ein Ablauf besteht aus drei Hauptbereichen.

Der erste Teil ist der Unterprogramm Typnamen.

Beispiele für Typennamen sind Sub (Aufruf einer primären Sektion bzw. eines Unterprogramms des primären Typs)

sowie `Files` und `WinBatch` (diese Aufrufe sind speziell für die zweite Sektion).

Den kompletten Überblick über die existierenden Sub-Programmtypen sind am Anfang von [Kapitel "Aufrufe von Unterprogrammen"](#) genauer beschrieben.

Der zweite Teil bestimmt, wo und wie die Zeilen des Subprogramms gefunden werden. Dazu gibt es zwei Möglichkeiten:

1. Das Sub-Programm ist eine Zeilenabfolge, die im sich ausführbaren Bereich des *opsi-winst/opsi-script* Skripts als interne Sektion befindet. Es wird ein eindeutiger Sektionsname (bestehend aus Buchstaben, Zahlen und einigen Sonderzeichen) hinzugefügt, um den Programmtyp näher zu beschreiben (ohne Leerzeichen).

z.B.

```
sub_install_winXP
```

oder

```
files_copy_winXP
```

Sektionsnamen sind case insensitive wie jeder andere String.

2. Wenn der Programmtyp alleine steht, wird eine String-Liste oder ein String-Ausdruck erwartet. Wenn der folgende Ausdruck nicht als String-Listenausdruck aufgelöst werden kann (vgl. 3.) wird ein String-Ausdruck erwartet. Der String wird dann als Dateiname interpretiert. Der *opsi-winst/opsi-script* versucht die Datei als Textdatei zu öffnen und interpretiert die Zeilen als eine externe Sektion des beschriebenen Typs.

Bsp.:

```
sub "p:\install\opsiutils\mainroutine.ins"
```

Es wird versucht die Zeile `mainroutine.ins` als Anweisung der Subsektion auszulesen.

Der String wird als Dateiname interpretiert und kann dabei folgendes sein:

- Ein kompletter Pfad zu einer Datei. [W/L]
- Eine Datei in `%ScriptPath%` [W/L]
- Eine Datei in `%opsiScriptHelperPath%\lib` [W]
Entspricht: `%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib`
- Eine Datei in `%ScriptPath%/../lib` [W/L]
- Eine Datei in `%WinstDir%\lib` [W]

Die Prüfung erfolgt in dieser Reihenfolge. Die erste Datei die gefunden wird, wird genommen.

1. Wenn der Ausdruck auf eine alleinstehenden spezifizierten Sektionstyp folgt, kann dieser als ein String-Listenausdruck aufgelöst werden. Die String-Listenkomponenten werden dann als ein Sektionsausdruck interpretiert.

Dieser Mechanismus kann bspw. dazu verwendet werden, um eine Datei mit Unicode-Format zu laden und dann mit den üblichen Mechanismen zu bearbeiten:

```
registry loadUnicodeTextFile("%scriptpath%/opsiorgkey.reg") /regedit
```

Syntaktisch hat diese Zeile die drei Bestandteile:

* `registry`, die eigentliche Anweisung, die den Sektionstyp spezifiziert.

* `loadUnicodeTextFile (..)`, ein String-Listenausdruck, in dem näher beschrieben ist, wie man eine Zeile der `registry` Sektion bekommt.

* `/regedit`, Option als 2. Parameter (typspezifisch, s. das Folgende).

In diesem Beispiel gibt der Aufrufparameter ein Beispiel an für den dritten Teil eines Subsektionsaufrufs:

Der dritte Part eine Aufrufs umfasst spezielle Aufrufsoptionen. Referenzen für die Aufrufsoptionen beziehungsweise für eine genauere Beschreibung der Sektionsaufrufe finden sich in siehe [Kapitel "Sekundäre Sektionen"](#).

9.17 Reboot-Steueranweisungen

Die Anweisung `ExitWindows` dient zur Steuerung von Reboot, Shutdowns, u.ä. Vorgängen welche erst nach Beendigung des *opsi-winst/opsi-script* selbst durchgeführt werden. Die Benennung des Befehls und die Tatsache, dass es `ExitWindows` nicht ohne Modifier gibt, ist historisch bedingt: Unter Windows 3.1 konnte man Windows beenden und zur DOS-Ebene zurück wechseln.

- `ExitWindows /RebootWanted`
Abgekündigt: vermerkt eine Rebootanfrage eines Skriptes in der Registry, lässt aber das *opsi-winst/opsi-script* Skript weiterlaufen und weitere Skripte abarbeiten und rebootet erst, wenn alle Skripte durchgelaufen sind. Eigentlich wird dieses Kommando jetzt als `ExitWindows /Reboot` behandelt (da ansonsten eine Installation fehlschlagen könnte, weil ein benötigtes Produkt nicht komplett installiert wurde).
- `ExitWindows /Reboot [W/L]`
unterbricht eine Skriptfolge durch die Auslösung des Reboots nachdem der *opsi-winst/opsi-script* die Bearbeitung des laufenden Skripts beendet hat.
- `ExitWindows /ImmediateReboot [W/L]`
unterbricht die normale Ausführung eines Skripts, an der Stelle, an der er aufgerufen wird. Nach dem der Befehl aufgerufen wurde, werden (außer `if`-Anweisungen) keine Anweisungen mehr ausgeführt und der Rechner rebootet. Dabei bleibt in der opsi-Umgebung der Actionrequest der das Skript aufgerufen hat bestehen. Dadurch wird gewährleistet, dass nach dem Neustart der *opsi-winst/opsi-script* wieder das Skript, das abgebrochen wurde, startet. Das Skript sollte daher so konzipiert sein, dass die Ausführung nach dem Punkt der Unterbrechung fortgesetzt wird (andernfalls erhalten wir möglicherweise eine Endlosschleife. . .) vgl. das Beispiel in diesem Abschnitt.
- `ExitWindows /ImmediateLogout [W]`
funktioniert ähnlich wie der Aufruf `/ImmediateReboot`, aber beinhaltet ein Beenden des *opsi-winst/opsi-script* (Beenden des Skriptes) statt einen Reboot. Dies ist dann sinnvoll, wenn nachfolgend ein automatisches Login (eines anderen Users) folgen soll. Beachten Sie hierzu [Kapitel "Skript für Installationen im Kontext eines lokalen Administrators"](#).
- `ExitWindows /ShutdownWanted [W]`
sorgt dafür, dass der PC nach Abschluss der Installation aller angefragten Produkte heruntergefahren wird.

Wie man eine Markierung setzt, um sicherzustellen, dass das Skript nicht in eine Endlosschleife läuft, wenn `ExitWindows /ImmediateReboot` aufgerufen wird, demonstriert folgendes Codebeispiel:

```
DefVar $Flag$
DefVar $WinstRegKey$
DefVar $RebootRegVar$

Set $WinstRegKey$ = "HKLM\SOFTWARE\opsi.org\winst"
Set $Flag$ = GetRegistryStringValue32("[+$WinstRegKey$+" "RebootFlag")

if not ($Flag$ = "1")
;=====
; Anweisungen vor Reboot

Files_doSomething

; Reboot initialisieren ...
Set $Flag$ = "1"
```

```

Registry_SaveRebootFlag
ExitWindows /ImmediateReboot

else
;=====
; Anweisungen nach Reboot

; Rebootflag zurücksetzen
Set $Flag$ = "0"
Registry_SaveRebootFlag

; die eigentlichen Anweisungen

Files_doMore

endif

[Registry_SaveRebootFlag]
openKey [$WinstRegKey$]
set "RebootFlag" = "$Flag$"

[Files_doSomething]
; eine Sektion, die vor dem Reboot ausgeführt wird

[Files_doMore]
; eine Sektion, die nach dem Reboot ausgeführt wird

```

9.18 Scriptabbruch und fehlgeschlagene Installation anzeigen [W/L]

Passieren bei einer Installation Fehler, die zum Fehlschlagen der Installation führen, so sollte dies an den Server zurückgemeldet werden.

Um in einem *opsi-winst/opsi-script* Skript, eine Installation als gescheitert zu erklären, gibt es eine Ausdruck namens `isFatalError` [W/L] unterbricht die normale Ausführung eines Skripts, an der Stelle, an der er aufgerufen wird. Nach dem der Befehl aufgerufen wurde, werden keine Anweisungen mehr ausgeführt und als Skriptergebnis wird *failed* zurückgeliefert. Wird dieser Befehl nicht aufgerufen, so ist das Skriptergebnis *success*.

Seit 4.11.3.2 ist auch die folgende Variante erlaubt:

- `isFatalError <string>` [W/L]
wobei `<string>` als kurze Fehlerbeschreibung an den opsi-server als *actionProgress* weitergegeben wird und im opsi-configed angezeigt wird.

Es gibt **keinen** Automatismus innerhalb eines Winst-Skriptes, um zu einen *failed* Ergebnis zu kommen. Sie müssen skriptgesteuert den Fehler selbst feststellen. Hierzu gibt Ihnen der *opsi-winst/opsi-script* einige Hilfsmittel.

Ein „fataler Fehler“ sollte zum Beispiel ausgelöst werden, wenn der Plattenplatz für die Installation nicht ausreicht:

```

DefVar $SpaceNeeded$
Set $SpaceNeeded$ = "200 MB"

if not(HasMinimumSpace ("%SYSTEMDRIVE%", $SpaceNeeded$))
  LogError "Nicht genügend Platz. Erforderlich sind "+$SpaceNeeded$
  isFatalError

```

```

; beendet die Skriptausführung und setzt den Produktstaus auf failed
else
; die Installation wird gestartet
; ...
endif

```

Fehler die von Funktionen des *opsi-winst/opsi-script* zurückgeliefert werden, werden in die Logdatei geschrieben und erhöhen den Fehlerzähler des opsi-winst. Dieser Fehlerzähler kann ausgewertet werden. So besteht auch die Möglichkeit, in einem kritischen Abschnitt eines Skripts festzustellen, ob Fehler bzw. wie viele Fehler aufgetreten sind (und abhängig hiervon ggf. `isFatalError` aufzurufen).

- `markErrorNumber` [W/L]
startet die Fehlerzählung für einen Abschnitt.
Die Zahl der Fehler, die ab dieser Stelle aufgetreten sind, kann dann abgerufen werden mit dem Ausdruck:
`errorsOccurredSinceMark`
- `errorsOccurredSinceMark` [W/L]
wird verwendet um das Auftreten von Fehlern seit dem Aufruf von `markErrorNumber` zu ermitteln. Dieser Ausdruck dient dann zum Vergleich mit einer Zahl.
Z.B. kann man die Bedingung „es kam in diesem Abschnitt mindestens ein Fehler vor“ so formulieren:
`if errorsOccurredSinceMark > 0`

Sofern die Skriptanweisungen nicht direkt einen Fehler produzieren, jedoch aufgrund bestimmter Umstände eine Situation trotzdem als Fehlersituation gewertet werden soll, kann auch mittels der Anweisung `logError` eine Fehlermeldung generiert werden.

```

markErrorNumber
; Fehler, die nach dieser Marke auftreten werden gezählt
; und werden als fatale Fehler gewertet

logError "test error"
; wir schreiben einen Kommentar "test error" in die Logdatei
; und die Fehleranzahl wird um eins erhöht
; für Testzwecke kann man diese Zeile auskommentieren

if errorsOccurredSinceMark > 0
; die Skriptausführung wird so bald wie möglich beendet
; und setzt den Produktstatus auf "failed"

isFatalError
; Kommentare können noch geschrieben werden

comment "error occured"

else
; kein Fehler aufgetreten, gibt folgendes aus:

comment "no error occured"
endif

```

- `isSuccess` //since 4.11.3.7 [W/L]
Bricht das Script ab ohne einen Fehler zu melden.
- `noUpdateScript` //since 4.11.3.7 [W/L]
Führt nach einem Setup kein Updatescript aus auch wenn eines vorhanden ist.

- `isSuspended` //since 4.11.4.1 [W/L] Bricht die Scriptarbeit ab ohne das ein Erfolg oder Fehler gemeldet werden. Der `ActionRequest` (z.B. `setup`) bleibt unverändert.

9.19 Lokale Funktionen [W/L]

Seit Version 4.12 kennt opsi-script auch lokale Funktionen.

Ein Beispiel:

```
DefFunc myFunc(val $str1$ : string, $str2$ : string) : string
    set $result$ = $str1$ + $str2$
endfunc
```

9.19.1 Konzept

Ziel dieser Erweiterung ist die Umsetzung folgender Konzepte:

- Funktionen mit Rückgabewert:

Die Funktionen haben einen Rückgabewert welche vom Typ `string` oder `stringlist` ist. Der Aufruf eine solchen Funktion kann überall da Erfolgen, wo ein Stringausdruck bzw.eine Stringliste erwartet wird.

Funktionen ohne Rückgabewert sind ab 4.12.0.16 auch erlaubt und müssen als `void` deklariert werden..
- Frei definierbare Aufrufparameter: Einer Funktion können Parameter übergeben werden. Diese Parameter werden bei der Deklaration der Funktion definiert. Die Aufrufparameter können vom Typ `string` oder `stringlist` sein. Die Aufrufparameter können als *CallByValue* oder per *callByReference* übergeben werden. *CallByValue* ist der Default. Das bedeutet: wird keine Aufrufmethode explizit angegeben, so wird *CallByValue* verwendet. Soll *CallByValue* explizit angegeben werden, so erfolgt dies über das Schlüsselwort `val`. *CallByValue* bedeutet, das beim Aufruf der Inhalt einer beim Aufruf verwendeten Variable auf die Aufrufvariable kopiert wird. *CallByReference* muß über das Schlüsselwort `ref` explizit angegeben werden. *callByReference* bedeutet, dass beim Aufruf eine Verbindung zwischen der aufrufenden Variablen und des lokalen Aufrufparameters erstellt wird. Eine Änderung der lokalen Variable des Aufrufparameters, wirkt sich direkt auf die beim Aufruf verwendete Variable aus.

Die Übergabeparameter stehen innerhalb der Funktion als lokale Variablen zur Verfügung.
- Lokale Variablen:

Eine Funktion enthält lokale Variablen. Implizit gibt es die Aufrufparameter als lokale Variablen und die Variable `$result$` welche vom Typ des Rückgabewertes ist. Darüberhinaus können weitere Variablen innerhalb der Funktion definiert werden.

All dies Variablen sind lokal, d.h. sie sind nur innerhalb dieser Funktion sichtbar. Eine lokale Variable mit dem selben Namen einer globalen Variable verdeckt innerhalb der Funktion die entsprechende globale Variable.
- Geschachtelte Funktionen:

Eine lokale Funktion kann wiederum eine oder mehrerr Definitionen von lokalen Funktionen enthalten. Diese Funktionen sind nur innerhalb der Funktion sichtbar in der sie definiert sind.
- Rekursive Aufrufe:

Eine Funktion kann sich selbst rekursiv aufrufen.
- Primäre und sekundäre Sektionen innerhalb von Funktionen:

Der Funktionskörper kann eigene Sektionen enthalten. Diese sind lokal zu dieser Funktion also nur innerhalb der Sektion sichtbar.

9.19.2 Syntax

Definition

```
DefFunc <func name>([calltype parameter ptype][,[calltype parameter ptype]]) : ftype
<function body>
endfunc
```

Dabei ist:

- `DefFunc` das Schlüsselwort zur Beginn der Definition einer lokalen Funktion.
- `<func name>` der frei gewählte Name der Funktion.
- `calltype` ist der Aufruftyp [`val` | `ref`]. Wird kein Aufruftyp angegeben, + so gilt `val` als gesetzt.
- `parameter` ist der freigewählte Name des Aufrufparameters, welcher unter diesem Namen innerhalb der Funktion als lokale Variable zur Verfügung steht.
- `ptype` ist der Datentyp des Parameters und entweder `string` oder `stringlist`;
- `ftype` ist der Datentyp der Funktion und entweder `string`, `stringlist` oder `void`. Die `void` Deklaration bedeutet, das es keine Rückabe gibt.
- `<function body>`: ist der Körper der Funktion, welcher dem opsi-script syntax genügen muß.
In diesem Teil gibt es die automatisch erzeugte lokale Variable `$result$`, welche den Datentyp der Funktion hat (String/Stringliste) und dazu dient den Rückgabewert aufzunehmen.
- `endfunc` zeigt als Schlüsselwort das Ende einer Funktionsdefinition an.

Eine Funktion muss definiert werden **bevor** sie aufgerufen werden kann.

9.19.3 Beispiele

Einfache Funktion welche zwei Strings miteinander verbindet:

```
[actions]
DefVar $mystr$
DefVar $str1$
set $str1$ = 'ha'

DefFunc myFunc(val $str1$ : string, $str2$ : string) : string
    set $result$ = $str1$ + $str2$
endfunc

set $mystr$ = myFunc("he", "ho")
set $mystr$ = myFunc("he", timeStampAsFloatStr)
set $mystr$ = myFunc("he", $str1$)
```

Erwartete Ergebnisse:

- `heho`
- `he42921.809`
- `heha`

Funktion vom Type `stringlist` welcher ein `string` und eine `stringlist` übergeben werden:

```
[actions]
DefVar $mystr$
DefVar $str1$
DefStringlist $list1$
DefStringlist $list2$

set $str1$ = 'ha'

DefFunc myFunc1(val $str1$ : string, $list1$ : stringlist) : stringlist
    set $result$ = createStringlist($str1$ , takeString(2,$list1$))
endfunc

set $list2$ = splitstring("/etc/opsi/huhu","/")
set $list1$ = myFunc1("hi",$list2$)
```

Erwartete Ergebnisse:

- \$list1\$ = [hi,opsi]

Funktion vom Type `string` welcher ein boolescher `string` übergeben wird:

```
[actions]

DefFunc myFunc2($str1$ : string) : string
    set $result$ = booltoststring($str1$)
endfunc

if stringtobool(myfunc2('1 > 0'))
    comment "true"
else
    comment "false"
endif
```

Erwartete Ergebnisse:

- *true*

Funktion vom Type `string` welcher ein `string` übergeben wird mit lokaler Variable:

```
[actions]
DefVar $mystr$

DefFunc myFunc3($str1$ : string) : string
    DefVar $locstr1$
    set $locstr1$ = '123'
    set $result$ = $locstr1$ + $str1$
endfunc

set $mystr$ = myFunc3("he")
```

Erwartete Ergebnisse:

- *123he*

Funktion vom Type `string` welcher ein `string` übergeben wird mit lokaler Variable und geschachtelter Funktion:

```
[actions]
DefVar $mystr$

DefFunc myFunc4($str1$ : string) : string
  DefVar $locstr1$

  DefFunc myFunc5($str1$ : string) : string
    set $result$ = 'inner' + $str1$
  endfunc

  set $locstr1$ = '123'
  set $result$ = $str1$ + myFunc5($locstr1$)
endfunc

set $mystr$ = myFunc4("outer")
```

Erwartete Ergebnisse:

- *outerinner123*

Einfache Funktion vom Type `string` welcher ein `string` by reference übergeben wird mit lokaler Variable:

```
[actions]
DefVar $mystr$
DefVar $str1$
DefVar $str2$

set $str1$ = 'ha'
set $str2$ = 'hi'

DefFunc myFunc6(ref $str1$ : string) : string
  DefVar $locstr1$
  set $locstr1$ = '123'
  set $str1$ = 'setinlocal'
  set $result$ = $locstr1$ + $str1$
endfunc

set $mystr$ = myFunc6($str2$)
set $mystr$ = $str1$ + $str2$
```

Erwartete Ergebnisse:

- *123setinlocal*
- *hasetinlocal*

Funktion vom Type `stringlist` welcher eine Variable vom Type `stringlist` mit *call by reference* übergeben wird mit lokalen `stringlist` Variable:

```
[actions]
DefVar $mystr$
DefStringlist $list1$
DefStringlist $list2$

et $list2$ = splitstring("/etc/opsi/huhu", "/")
```

```

DefFunc myFunc7(ref $list1$ : stringlist) : stringlist
  DefStringlist $loclist1$
  set $loclist1$ = splitstring("/a/b/c","/")
  set $list1$ = createStringList('setinlocal')
  set $loclist1$ = addListToList($loclist1,$list1$)
  set $result$ = $loclist1$
endfunc

set $list1$ = myFunc7($list2$)
comment "$list2$ index 0: " + takestring(0,$list2$)

```

Erwartete Ergebnisse:

- \$list1\$ = [a,b,c,setinlocal]
- *setinlocal*

Funktion vom Type `stringlist` welcher ein `string` übergeben wird mit lokaler Variable und lokaler sekundärer Sektion:

```

[actions]
DefStringlist $list1$

DefFunc myFunc8($str1$ : string) : stringlist
  DefStringlist $loclist1$
  set $loclist1$ = getoutstreamfromsection("shellInAnIcon_test")
  set $result$ = $loclist1$

  [shellinanon_test]
  set -x
  $str1$
endfunc

set $list1$ = myFunc8('pwd')

```

Erwartete Ergebnisse:

- \$list1\$ = [+ pwd, /home/uib/gitwork/lazarus/opsi-script]

Funktion vom Type `void` (also ohne Rückgabe) welcher ein `string` übergeben wird:

```

[actions]
ScriptErrorMessages = false
DefVar $str1$

set $str1$ = 'haha'

DefFunc myNoResultFunc(ref $str1$ : string) : void
  set $str1$ = "huhu"
endfunc

myNoResultFunc($str1$)
comment "$str1$ is: "+$str1$

```

Erwartete Ergebnisse:

- \$str1\$ is: huhu

Funktion vom Type `string` welcher kein Parameter übergeben wird:

```
[actions]
ScriptErrorMessages = false
DefVar $str1$

DefFunc myNoParamFunc() : string
    set $result$ = "huhu"
endfunc

set $str1$ = myNoParamFunc()
```

Erwartete Ergebnisse:

- `$str1$` is: huhu

9.20 Import von Library Funktionen [W/L]

```
importLib <string expr> ; import library // since 4.12.0.0
```

```
<string expr> : <file name>[.<file extension>][::<function name>]
```

Wenn keine *.<file extension>* (Dateierweiterung) übergeben wird, so wird `.opsiscript` als Default verwendet.

Wenn kein *::<function name>* über geben wird, so werden alle Funktionen der angegebenen Datei importiert.

<file name> ist:

- Ein kompletter Pfad zu einer Datei. [W/L]
- Eine Datei in `%ScriptPath%` [W/L]
- Eine Datei in `%opsiScriptHelperPath%\lib` [W]
Entspricht: `%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib`
- Eine Datei in `%ScriptPath%/. /lib //since 4.11.5.2` [W/L]
- Eine Datei in `%WinstDir%\lib` [W] oder `/usr/share/opsi-script/lib` [L]

Die Prüfung erfolgt in dieser Reihenfolge. Die erste Datei die gefunden wird, wird genommen.

Kapitel 10

Sekundäre Sektionen

Sekundäre Sektionen können ebenso wie die primäre Sektion aufgerufen werden, haben aber eine andere Syntax. Die Syntax der verschiedenen Sektionstypen ist jeweils aufgabenbezogen und lehnt sich möglichst eng an bekannte Kommandoformen für den jeweiligen Aufgabentyp an.

Sekundäre Sektionen sind spezifiziert für einen eingegrenzten Funktionsbereich. Dies bezieht sich auf das Objekt der Funktion z.B. das Dateisystem im Allgemeinen, die Windows Registry oder XML-Dateien. Eine spezielle Bedeutung haben die verschiedenen Varianten der Batch-Sektionen, die dazu dienen (beliebige) externe Programme oder Skripte aufzurufen.

10.1 Files-Sektionen [W/L]

Files-Sektionen dienen zum Dateihandling (Kopieren, Löschen). Anders als bei Ausführung der vergleichbaren Kommandozeilen-Befehle, werden die ausgeführten Operationen bei Bedarf genau protokolliert. Zusätzlich kann beim Kopieren von Bibliotheksdateien (z.B. dll-Dateien) auch die Dateiversion geprüft werden, so dass nicht neuere Versionen durch ältere Versionen überschrieben werden.

10.1.1 Beispiele

Eine Files-Sektion könnte etwa lauten:

```
[Files_do_some_copying]
copy -sV "p:\install\instnsc\netscape\*.*" "C:\netscape"
copy -sV "p:\install\instnsc\windows\*.*" "%SYSTEMROOT%"
```

Mit diesen Anweisungen werden alle Dateien des Verzeichnisses `p:\install\instnsc\netscape` in das Verzeichnis `C:\netscape` kopiert, sowie alle Dateien von `p:\install\instnsc\windows` in das Windows-Systemverzeichnis (welches das jeweils Gültige ist, steht automatisch in der opsi-winst-Konstante `%SYSTEMROOT%`). Die Option `-s` bedeutet dabei, dass alle Subdirectories mit erfasst werden, `-V` steht für das Einschalten der Versionskontrolle von Bibliotheksdateien.

10.1.2 Aufrufparameter (Modifizier) [W]

In der Regel benötigt eine Files-Sektion beim Aufruf keinen Parameter.

Es gibt jedoch eine spezielle Anwendung der Files-Sektion, bei der Zielpfad von Kopieraktionen automatisch bestimmt oder modifiziert wird, bei denen die betreffende Files-Sektion mit dem Parameter

- `/AllNTUserProfiles` bzw.

- /AllNTUserSendTo

aufgerufen wird.

Beide Varianten bedeuten:

Die betreffende Files-Sektion wird je einmal für jeden NT-User ausgeführt. Bei Kopieraktionen in dieser Files-Sektion wird automatisch ein User-spezifisches Verzeichnis als Zielverzeichnis eingesetzt.

Für alle anderen Aktionen steht eine Variable %UserProfileDir% oder seit *opsi-winst/opsi-script* version 4.11.2 %CurrentProfileDir% zur Verfügung, mit der Verzeichnisnamen konstruiert werden können.

Das User-spezifische Verzeichnis ist im Fall von /AllNTUserProfiles das User-Profilverzeichnis. Im Fall von /AllNTUserSendTo wird der Pfad zum User-spezifischen SendTo-Ordner vorgegeben (der dazu dient, im Windows-Explorer Kontextmenü-Verknüpfungen vorzugeben).

Die genaue Regel, nach der Zielpfade für copy-Anweisungen automatisch gebildet werden, ist dreiteilig:

1. Folgt auf die Angabe der zu kopierenden Quelldateien gar keine Zielverzeichnisangabe, so werden die Dateien direkt in das betreffende User-spezifische Verzeichnis kopiert. Der Befehl lautet einfach


```
copy <Quelldatei>
```

 Dies ist gleichbedeutend mit


```
copy <Quelldatei> "%UserProfileDir%"
```

 oder seit 4.11.2


```
copy <source file(s)> "%CurrentProfileDir%"
```
2. Wenn außer den zu kopierenden Quelldateien ein Kopierziel *targetdir* spezifiziert ist, dieses Ziel jedoch keinen absoluten Pfad (beginnend mit Laufwerksname oder mit "\") darstellt, dann wird die Zielangabe als Unterverzeichnisname des User-spezifischen Verzeichnisses interpretiert und der Zielverzeichnisname dementsprechend konstruiert. D.h. man schreibt


```
copy <Quelldateien> targetdir
```

 und das wird interpretiert wie:


```
copy <Quelldatei> "%UserProfileDir%\targetdir"
```

 oder seit 4.11.2


```
copy <source file(s)> "%CurrentProfileDir%\targetdir"
```

Die Verwendung von %CurrentProfileDir% hat den Vorteil, das die selbe *Files* Sektion mit /AllNTUserProfiles verwendet werden kann wenn das Script nicht als *userLoginScript* (in *Machine* script mode) läuft und ohne /AllNTUserProfiles wenn das Script als *userLoginScript* läuft (in *Login* script mode).

1. Enthält der copy-Befehl Quelldateien und einen absoluten Zielpfad *targetdir*, so wird dieser statische Zielpfad verwendet.

Weiterhin gibt es die Aufrufparameter:

- /32Bit (Default)
- /64Bit
- /SysNative

Welche die *file redirection* auf 64 Bit-Systemen beeinflussen. siehe [Kapitel 64 Bit-Unterstützung](#)

10.1.3 Kommandos [W/L]

Innerhalb einer Files-Sektion sind die Anweisungen

- Copy [W/L]
- Delete / Del [W/L]
- SourcePath

- `CheckTargetPath` [W/L]
- `chmod` [L]
- `hardlink` [W/L]
- `symlink` [W/L]
- `rename` [W/L]
- `move` [W/L]

definiert.

Die Kommandos `Copy` und `Delete` entsprechen im Wesentlichen den Windows-Kommandozeilen-Befehlen `xcopy` bzw. `del`.

`SourcePath` sowie `CheckTargetPath` legen Quell- bzw. Zielpfad einer Kopieraktion fest, ähnlich wie sie etwa im Windows-Explorer durch Öffnen von Quell- und Zieldirectory in je einem Fenster realisiert werden kann. Der Zielpfad wird, sofern er nicht existiert, erzeugt.

Im Einzelnen:

- `Copy [-svdunxwnr] <Quelldatei(maske)> <Zielpfad>`

Die Quelldateien können dabei mittels Joker ("*" in der Dateimaske) oder auch nur mit einer Pfadangabe bezeichnet werden.



Achtung

Zielpfad wird in jedem Fall als Directory-Name interpretiert. Umbenennen beim Kopieren ist nicht möglich: Ziel ist immer ein Pfad, nicht ein (neuer) Dateinamen. Existiert der Pfad nicht, wird er (auch mit geschachtelten Directories) erzeugt.

Die einzelnen (in beliebiger Reihenfolge ausführbaren) Optionen der `Copy`-Anweisung bedeuten:

- `s` → Mit Rekursion in Subdirectories. [W/L]
- `e` → Leere (Empty) Subdirectories. [W]
Gibt es leere Subdirectories im Quellverzeichnis, werden sie im Zielverzeichnis ebenfalls leer ("empty") erzeugt.
- `v` → Mit Versionskontrolle [W]
Mit Versionskontrolle:
Neuere Versionen von Windows-Bibliotheksdateien im Zielverzeichnis werden nicht durch ältere Versionen überschrieben (bei unklaren Verhältnissen wird in einem Log-Eintrag gewarnt).
- `v` → (nicht Verwenden) [W]
Mit Versionskontrolle:
Veraltet; bitte nicht bei Betriebssystemversionen höher als Win2k verwenden, da hier nicht nur gegen das Zielverzeichnis, sondern auch gegen %SYSTEM% geprüft wird. Verwenden Sie stattdessen `-V`.
- `d` → Mit Datumskontrolle: [W]
Jüngere *.EXE-Dateien werden nicht durch ältere überschrieben.
- `u` → Datei-Update: [W]
Es werden Dateien nur kopiert, sofern sie nicht mit gleichem oder jüngerem Datum im Zielpfad existieren.
- `x` → x-tract [W]
Wenn eine Datei ein Zip-Archiv ist, wird es entpackt (x-tract). Vorsicht: Zip-Archive verbergen sich unter verschiedenen Dateinamen (z.B. sind Java jar-Dateien auch Zip-Archive), daher sollte man die Extract-Option nicht unbesehen auf alle Dateien anwenden. Achtung: Es werden keine Pfade entpackt.
- `w` → weak [W]
Dateien werden nur überschrieben, wenn sie keinen Schreibschutz haben (das Überschreiben ist "weak" (relativ schwach) im Vergleich zum Defaultverhalten, dem Ignorieren des Schreibschutzes).

- **n** → no over write [W]
Dateien werden nicht überschrieben.
- **c** → continue [W]
Wenn eine Systemdatei in Benutzung ist, kann sie erst nur nach einem Reboot überschrieben werden. Das *opsi-winst/opsi-script* default-Verhalten ist dabei, dass ein Datei in Benutzung zum Überschreiben beim nächsten Reboot markiert wird UND die *opsi-winst/opsi-script* Reboot Markierung gesetzt wird. Das Setzen der Option **-c** stellt den automatischen Reboot aus. Das Kopieren wird in diesem Fall erst dann vervollständigt, wenn ein Reboot auf eine andere Weise ausgelöst wird.
- **r** → read-only Attribute [W]
Nur wenn diese Option gesetzt ist, bleibt ein eventuell vorhandenes read-only-Attribut erhalten (im Gegensatz zu dem default-Verhalten, welches read-only Attribute ausschaltet).
- **h** → follow symlinks [L] //since 4.11.6.14
Unter Linux wird symbolischen Links auf Dateien und Verzeichnissen gefolgt. Es wird als nicht der symbolische Link sondern dessen Ziel kopiert.
- **Delete** [-sfcd[n]] <Pfad> [W/L]

oder

- **Delete** [-sfcd[n]] <Datei(maske)> [W/L]
Löschen einer Datei bzw. eines Verzeichnisses. Mögliche Optionen (die in beliebiger Reihenfolge aufgeführt sein können) sind:
 - **s** → subdirectories
Steht für die Rekursion in Subdirectories, das heißt, der ganze Pfad bzw. alle der Dateimaske entsprechenden Dateien im Verzeichnisbaum ab der angegebenen Stelle werden gelöscht.

Achtung

Der Befehl

```
delete -s c:\opsi
```



Bedeutet nicht lösche das Verzeichnis *c:\opsi* rekursiv, sondern lösche ab *c:* rekursiv alle Dateien namens *opsi* (und führt damit evtl. zum kompletten Durchsuchen der Festplatte). Zum rekursiven Löschen von *c:\opsi* verwenden Sie das Kommando:

```
delete -s c:\opsi\
```

Durch den angehängten Backslash ist deutlich, dass Sie ein Verzeichnis meinen.

Es ist sicherer das Kommando `del` stattdessen zu verwenden

- **f** → force
Erzwingt ("force") das Löschen auch von read-only-Dateien.
- **c** → continue [W]
Wenn eine Datei in Benutzung ist, kann sie im Rahmen eines Reboots gelöscht werden. Das *opsi-winst/opsi-script* default-Verhalten ist dabei, dass ein Datei in Benutzung zum Löschen beim nächsten Reboot markiert wird UND die *opsi-winst/opsi-script* Reboot Markierung gesetzt wird. Das Setzen der Option **-c** stellt den automatischen Reboot aus. Das Löschen wird in diesem Fall erst dann vervollständigt, wenn ein Reboot auf eine andere Weise ausgelöst wird.
- **d** [**n**] → date
Dateien werden nur gelöscht, sofern sie mindestens *n* Tage alt sind. Default für *n* ist 1.
- **del** [Options] <path[/mask]] //since 4.11.2.1 [W/L]
Arbeitet wie `delete` aber bei

```
del -s -f c:\not-exists
```

wenn *c:\not-exists* nicht existiert wird nicht das komplette *c:* nach `not-exists` durchsucht.

Beispiel (**Der anhängende Backslash darf weggelassen werden**):

```
del -sf c:\delete_this_dir
```

Anmerkung

+ Mit `del` oder `delete` lassen sich bestimmte Verzeichnisse aus Sicherheitsgründen nicht löschen:

```
c:\,c:\windows,c:\windows\system32,*
```

- **SourcePath** = <Quelldirectory>
Festlegung des Verzeichnisses <Quelldirectory> als Vorgabe-Quelldirectory für in der betreffenden Sektion folgende Copy- sowie (!) Delete-Aktionen.
- **CheckTargetPath** = <Zieldirectory> [W/L]
Festlegung des Verzeichnisses <Zieldirectory> als Vorgabe-Zieldirectory für Copy-Aktionen. Wenn <Zieldirectory> nicht existiert, wird der Pfad auch mehrstufig erzeugt.
- **chmod** <mask> <path> //since 4.11.4.1 [L]
Setzt die Rechte von <path> auf <mode>. <mode> ist dabei die Numerische (Octale) angebe der Rechte (z.B. "755").
- **hardlink** <existing file> <new file> // since 4.11.5 [W/L]
Ein existierender <new file> wird überschrieben.
hardlink funktioniert nur auf Filesystemen die Hardlinks unterstützen wie NTFS und Standard Linux Filesysteme.
- **symlink** <existing file> <new file> // since 4.11.5 [W/L]
Unter Windows ist **symlink** erst ab NT6 und aufwärts verfügbar !
Ein existierender <new file> wird überschrieben.
- **rename** <old filename> <new filename> // since 4.11.5 [W/L]
move <old filename> <new filename> // since 4.11.5 [W/L]
Es gibt keine Unterschiede zwischen **rename** und **move**, es sind zwei Namen für die selbe Funktion
Ein existierender <new file> wird überschrieben.
Derzeit können keine Verzeichnisse umbenannt / verschoben werden.

Windows: <new filename> darf in einem anderen Directory liegen oder auch in einem anderen Volume / Disk. Im zweiten Fall, wird die Datei kopiert und danach das Original gelöscht.

Läßt sich das Ziel nicht erstellen (Datei in Verwendung) so wird die Operation beim nächsten Reboot fertiggestellt. Das funktioniert natürlich nur wenn das Zielfilesystem zum Reboot-Zeitpunkt verfügbar ist, also nicht auf Netzwerkshares. In diesem Fall wird auch automatisch ein Reboot nach dem Ende des Scriptes ausgelöst. Dies läßt sich aber mit der Option `-c` (continue) unterdrücken.

Die Erstellung von Junctions unter Windows wird noch nicht unterstützt.

Linux: <new filename> darf in einem anderen Directory liegen aber nicht in einem anderen Filesystem / Partition. Die Option `-c` wird unter Linux ignoriert.

Example:

```
[Files_link_move]
hardlink "$HomeTestFiles$\files\dummy.txt" "$HomeTestFiles$\files\hardlink.txt"
symlink "$HomeTestFiles$\files\dummy.txt" "$HomeTestFiles$\files\symlink.txt"
rename "$HomeTestFiles$\files\temp\dummy2.txt" "$HomeTestFiles$\files\temp\rename.txt"
move "$HomeTestFiles$\files\temp\dummy2.txt" "$HomeTestFiles$\files\temp\move.txt"
```

10.2 Patches-Sektionen [W/L]

Eine Patches-Sektion dient der Modifikation (dem "Patchen") einer "*.INI-Datei", d.h. einer Datei, die Sektionen mit Einträgen der Form <Variable> = <Wert> besteht. Die Sektionen oder Abschnitte sind dabei gekennzeichnet durch Überschriften der Form [Sektionsname].

(Seitdem eine gepatchete INI-Datei auf die gleiche Weise wie die Sektionen vom *opsi-winst/opsi-script* Skript erstellt werden, muss man vorsichtig mit den Bezeichnungen umgehen, damit kein Durcheinander entsteht).

10.2.1 Beispiele

```
Patches_DUMMY.INI $HomeTestFiles$\dummy.ini"
```

```
[Patches_dummy.ini]
add [secdummy] dummy1=add1
; werden durch andere Funktionen ueberschrieben
add [secdummy] dummy2=add2
add [secdummy] dummy3=add3
add [secdummy] dummy4=add4
add [secdummy] dummy5=add5
add [secdummy] dummy6=add6
set [secdummy] dummy2=set1
addnew [secdummy] dummy1=addnew1
change [secdummy] dummy3=change1
del [secdummy] dummy4
Replace dummy6=add6 replace1=replace1
```

ergibt folgenden Log:

```
Execution of Patches_DUMMY.INI
  FILE C:\tmp\testFiles\dummy.ini
  Info: This file does not exist and will be created
addEntry [secdummy] dummy1=add1
  addSection [secdummy]
  done
  done
addEntry [secdummy] dummy2=add2
  done
addEntry [secdummy] dummy3=add3
  done
addEntry [secdummy] dummy4=add4
  done
addEntry [secdummy] dummy5=add5
  done
addEntry [secdummy] dummy6=add6
  done
setEntry [secdummy] dummy2=set1
  Entry dummy2=add2
  changed to dummy2=set1
addNewEntry [secdummy] dummy1=addnew1
  appended entry
changeEntry [secdummy] dummy3=change1
  entry dummy3=add3
  changed to dummy3=change1
delEntry [secdummy] dummy4
  in section secdummy deleted dummy4=add4
replaceEntrydummy6=add6 replace1=replace1
  replaced in line 7
C:\tmp\testFiles\dummy.ini saved back
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich `$Flag_winst_patches$ = "on"`

10.2.2 Aufrufparameter

Der Name der zu patchenden Datei wird als Parameter übergeben.

Als optionalen Modifier gibt es:

- **/AllNTUserProfiles**

Wird eine Patches Sektion mit diesem Modifier aufgerufen und der Pfad zur zu patchenden Datei enthält die Konstante `%UserProfileDir%`, so wird diese Patchsektion für alle Profile ausgeführt.

Eine *Patches* Sektion welche in einer `[ProfileActions]` Sektion aufgerufen wird hat im *Machine* Modus den Modifier `/AllNTUserProfiles` implizit. Im Loginscript Modus wird dann `%UserProfileDir%` als `%CurrentProfileDir%` interpretiert.

(Seit Version 4.11.3.2)

10.2.3 Kommandos

In einer Patches-Sektion sind die Anweisungen

- `add`
- `set`
- `addnew`
- `change`
- `del`
- `delsec`
- `replace`

definiert. Eine Anweisung bezieht sich jeweils auf eine Sektion der zu patchenden Datei. Der Name dieser Sektion steht in Klammern `[]`.

Syntax und Funktion der Anweisungen im Einzelnen:

- **add** [`<section name>`] `<variable1> = <value1>`
Fügt einen Eintrag der Form `<Variable1> = <value1>` in die Sektion `<section name>` ein, falls dort noch kein Eintrag von `<Variable1>` (auch mit anderem Wert) existiert. Im anderen Fall wird nichts geschrieben. Existiert die Sektion noch nicht, wird sie zuerst erzeugt.
- **set** [`<section name>`]`<variable1> = <value1>`
Setzt einen vorhandenen Eintrag `<variable1> = <value X>` in der Sektion `<section name>`, um auf `<variable1> = <value1>` zu kommen. Existieren mehrere Einträge von `<variable1>`, wird der Erste umgesetzt. Falls kein Eintrag mit `<variable1>` existiert, wird `<variable1> = <value1>` in der Sektion `<section name>` erzeugt; existiert die Sektion noch nicht, wird sie zuerst erzeugt.
- **addnew** [`<section name>`]`<variable1> = <value1>`
Der Eintrag `<variable1> = <value1>` wird in der Sektion `<section name>` auf jeden Fall erzeugt, sofern er dort nicht bereits genau so existiert (gegebenenfalls zusätzlich zu anderen Einträgen von `<variable1>`). Existiert die Sektion noch nicht, wird sie zuerst erzeugt.
- **change** [`<section name>`]`<variable1> = <value1>`
Ändert einen vorhandenen Eintrag von `<variable1>` in der Sektion `<section name>` auf `<variable1> = <value1>`. Falls `<variable1>` nicht vorhanden ist, wird nichts geschrieben.
- **del** [`<section name>`] `<variable1> = <value1>`
bzw.
del [`<section name>`] `<variable1>`
In der Sektion `<section name>` wird gemäß dem ersten Syntaxschema der Eintrag `<variable1> = <value1>` entfernt. Nach dem zweiten Syntaxschema wird der erste Eintrag von `<variable1>` aus der angesprochenen Sektion gelöscht, unabhängig vom Wert des Eintrags.

- **delsec** [<section name>]
Die Sektion <section name> der .INI-Datei wird mitsamt ihren Einträgen gelöscht.
- **replace** <variable1>=<value1> <variable2>=<value2>
In allen Sektionen der .INI-Datei wird der Eintrag <variable1>=<value1> durch <Variable2>=<value2> ersetzt. Zur Anwendung dieses Befehls dürfen Leerzeichen weder um die Gleichheitszeichen stehen noch in <value1> bzw. <value2> enthalten sein.

10.3 PatchHosts-Sektionen [W/L]

Eine PatchHosts-Sektion dient der Modifikation einer *hosts*-Datei, das heißt einer Datei, deren Zeilen nach dem Schema *ipAdresse Hostname Aliasname(n) # Kommentar* aufgebaut sind.

Dabei sind *Aliasname(n)* und *Kommentar* optional. Eine Zeile kann auch mit dem Symbol *#* beginnen und ist dann insgesamt ein Kommentar.

Die zu patchende Datei kann als Parameter des **PatchHosts**-Aufrufs angegeben sein. Fehlt der Parameter *HOSTS*, so wird in den Verzeichnissen (in dieser Reihenfolge) *c:\nfs*, *c:\windows* sowie *%systemroot%\system32\drivers* etc nach einer Datei mit dem Namen *hosts* gesucht.

Wird auf keine dieser Arten eine Datei mit dem Namen *hosts* gefunden, bricht die Bearbeitung mit einem Fehler ab.

In einer PatchHosts-Sektion existieren die Anweisungen

- **setAddr**
- **setName**
- **setAlias**
- **delAlias**
- **delHost**
- **setComment**

Beispiel:

```
PatchHosts_add $HomeTestFiles$\hosts"
```

```
[PatchHosts_add]
setAddr ServerNo1 111.111.111.111
setName 222.222.222.222 ServerNo2
setAlias ServerNo1 myServerNo1
setAlias 222.222.222.222 myServerNo2
setComment myServerNo2 Hallo Welt
```

ergibt folgenden Log:

```
Execution of PatchHosts_add
  FILE C:\tmp\testFiles\hosts
  Set ipAddress 111.111.111.111 Hostname "ServerNo1"
  Set Hostname "ServerNo2" for ipAddress 222.222.222.222
  Alias "myServerNo1" set for entry "ServerNo1"
  Alias "myServerNo2" set for entry "222.222.222.222"
  SetComment of Host "myServerNo2" to "Hallo Welt"
  C:\tmp\testFiles\hosts saved back
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich `$Flag_winst_patch_hosts$ = "on"`.

Die Anweisungen im einzelnen:

- **setaddr** <hostname> <ipadresse>
Setzt die IP-Adresse für den Host <hostname> auf <ipadresse>. Falls noch kein Eintrag für den Host <hostname> besteht, wird er neu eingetragen.
- **setname** <ipadresse> <hostname>
Setzt den Namen des Hosts mit der angegebenen IP-Adresse <ipadresse> auf <hostname>. Falls noch kein Eintrag mit der IP-Adresse <ipadresse> existiert, wird er neu erzeugt.
- **setalias** <hostname> <alias>
Fügt für den Host mit dem IP-Namen <hostname> einen ALIAS-Namen <alias> ein.
- **setalias** <IPadresse> <alias>
Fügt für den Host mit der IP-Adresse <IPadresse> einen ALIAS-Namen <alias> ein.
- **delalias** <hostname> <alias>
Löscht aus dem Eintrag für den Host mit dem IP-Namen <hostname> den ALIAS-Namen <alias>.
- **delalias** <IPadresse> <alias>
Löscht aus dem Eintrag für den Host mit der IP-Adresse <IPadresse> den ALIAS-Namen <alias>.
- **delhost** <hostname> Löscht den Eintrag des Hosts mit dem IP- (oder Alias-) Namen <hostname>.
- **delhost** <IPadresse>
Löscht den Eintrag des Hosts mit der IP-Adresse <IPadresse>.
- **setComment** <ident> <comment>
Setzt für den Host mit dem IP-Namen, Alias-Namen oder Adresse <ident> den Kommentareintrag auf <comment>.

10.4 IdapiConfig-Sektionen

Eine IdapiConfig-Sektion waren dazu geeignet, in *idapi*.cfg*-Dateien, die von der Borland-Database-Engine verwendet werden, die benötigten Parameter einzufügen.

IdapConfig-Sektionen werden vom aktuellen *opsi-winst/opsi-script* nicht mehr unterstützt.

10.5 PatchTextFile-Sektionen [W/L]

PatchTextFile-Sektionen dienen zum Patchen allgemeiner Textdateien. Es gibt aber auch Spezialanweisungen zum Patchen von Mozilla Konfigurationsdateien.

Wichtig für die Arbeit mit Textdateien ist das Überprüfen, ob eine bestimmte Zeile bereits in einer existierenden Datei vorhanden ist. Für diese Zweck gibt es die boolesche Funktionen `Line_ExistsIn` und `LineBeginning_ExistsIn` (vgl. [Kapitel "Boolesche Ausdrücke"](#)) zur Verfügung.

10.5.1 Aufrufparameter

Der Name der zu patchenden Datei wird als Parameter übergeben.

Als optionalen Modifier gibt es:

- **/AllNTUserProfiles**

Wird eine *PatchTextFile* Sektion mit diesem Modifier aufgerufen und der Pfad zur zu patchenden Datei enthält die Konstante `%UserProfileDir%`, so wird diese Patchsektion für alle Profile ausgeführt.

Eine *PatchTextFile* Sektion welche in einer [ProfileActions] Sektion aufgerufen wird hat im *Machine* Modus den Modifier `/AllNTUserProfiles` implizit. Im Loginscript Modus wird dann `%UserProfileDir%` als `%CurrentProfileDir%` interpretiert.

(Seit Version 4.11.3.5)

10.5.2 Kommandos

Zwei Anweisungen dienen speziell dem komfortablen Patchen von Mozilla-Präferenzdateien:

- **Set_Mozilla_Pref** ("`<preference type>`", "`<preference key>`", "`<preference value>`")
sorgt dafür, dass in die beim Sektionsaufruf spezifizierten Datei die Zeile `<Präferenzvariable>` nach `<Wert>` geschrieben wird. Die ASCII-alphabetische Anordnung der Datei wird beibehalten bzw. hergestellt.
In den momentanen Mozilla Präferenzdateien gibt es folgende Ausdrücke
`user_pref("<key>", "<value>")`
`pref("<key>", "<value>")`
`lock_pref("<key>", "<value>")`
`defaultPref("<key>", "<value>")`
`lock_pref("<key>", "<value>")`
`clearPref("<key>", "<value>")`
Jeder dieser Ausdrücke, sozusagen jede (javascript) Funktion, die auf diese Weise aufgerufen wird
`functionname (String1, String2)`
kann mit diesem Kommando gepatcht werden über das Setzen des entsprechenden Strings für `<preference type>` (das ist bspw. für `functionname`).
Wenn der Starteintrag `"functionname (String1)"` in dem bearbeitenden File existiert, kann er gepatcht werden (und bleibt an seinem Platz). Andernfalls wird eine neue Zeile eingefügt.
Für den *opsi-winst/opsi-script* - ungewöhnlicherweise - sind alle Strings case sensitive.
- **Set_Netscape_User_Pref** ("`<Präferenzvariable>`", "`<Wert>`") ist die restriktivere, ältere Version des vorherigen Kommandos und sollte nicht mehr verwendet werden.
Setzen der Zeile mit den vom User vergebenen Präferenzen für die Variable `<Präferenzvariable>` und des Wertes `<value>`. (Abgekündigt!)
- **AddStringListElement_To_Mozilla_Pref** ("`<Präferenztyp>`", "`<Präferenzvariable>`", "`<add value>`")
fügt ein Element zu einem Listeneintrag der Präferenzdatei hinzu. Das Element wird überprüft, wenn der Wert, der hinzugefügt werden soll, bereits in der Liste existiert (dann wird er nicht hinzugefügt).
- **AddStringListElement_To_Netscape_User_Pref** ("`<Präferenzvariable>`", "`<Werteliste>`")
ist die restriktivere, ältere Version des vorherigen Kommandos und sollte nicht mehr verwendet werden.
Es fügt einer Werteliste ein Element hinzu (soweit nicht schon enthalten). Angewendet werden kann die Anweisung zur Ergänzung der No-Proxy-Einträge in der `prefs.js`. (Abgekündigt!)

Alle übrigen Anweisungen von PatchTextFile-Sektionen sind nicht auf spezielle Dateiararten bzw. eine spezielle Syntax der Datei festgelegt:

Die drei Suchanweisungen

- **FindLine** `<Suchstring>`
Findet eine Zeile die dem `<Suchstring>` komplett entspricht.
- **FindLine_StartingWith** `<Suchstring>`
Findet eine Zeile die mit `<Suchstring>` anfängt.
- **FindLine_Containing** `<Suchstring>`
Findet eine Zeile die `<Suchstring>` enthält.

durchsuchen die Datei ab der Position, auf der der Zeilenzeiger steht. Sofern sie eine passende Zeile finden, setzen sie den Zeilenzeiger auf die erste Zeile, die <Suchstring> gleicht / mit ihm beginnt / ihn enthält. Die Suche ist nicht case-sensitive.

Wird <Suchstring> nicht gefunden, so bleibt der Zeilenzeiger an der Ausgangsposition stehen.

- **GoToTop**
setzt den Zeilenzeiger vor die erste Zeile setzt (werden Zeilen gezählt muss man berücksichtigen, dass dieses Kommando den Zeilenzeiger über die Anfangszeile setzt). Der Zeilenzeiger kann vor und zurück bewegt werden mit der <Anzahl Zeilen>.
- **AdvanceLine** [<Anzahl Zeilen>]
bewegt den Zeilenzeiger um <Anzahl Zeilen> vor oder zurück.
- **GoToBottom**
setzt den Zeilenzeiger auf die letzte Zeile.
- **DeleteTheLine**
löscht die Zeile auf der der Zeilenzeiger steht, sofern sich dort eine Zeile befindet (wenn der Zeilenzeiger oben platziert ist, wird nichts gelöscht).
- **AddLine_** <Zeile> oder **Add_Line_** <Zeile>
<Zeile> wird am Schluss der Datei angehängt.
- **InsertLine** <Zeile> oder **Insert_Line_** <Zeile>
<Zeile> wird an der Stelle eingefügt, an der der Zeilenzeiger steht.
- **AppendLine** <Zeile> oder **Append_Line** <Zeile>
<Zeile> wird nach der Zeile eingefügt, an der der Zeilenzeiger steht.
- **Append_File** <Dateiname>
liest die Zeilen der Datei <Dateiname> ein und fügt sie an den Schluss der gerade bearbeiteten Datei an.
- **Subtract_File** <Dateiname>
entfernt die Anfangszeilen der bearbeiteten Datei, so weit sie mit den Anfangszeilen der Datei <Dateiname> übereinstimmen.
- **SaveToFile** <Dateiname>
speichert die bearbeitete Datei als <Dateiname>.
- **Sorted**
bewirkt, dass die Zeilen alphabetisch (nach ASCII) geordnet sind.
- **setKeyValueSeparator** <separator char> //since 4.11.4.4
setzt für key/value Paare (Befehl **setValueByKey**) das Trennzeichen (Default ist =)
- **setValueByKey** <keystr> <valuestr> //since 4.11.4.4
sucht ein key/value Paar mit dem key <keystr> und setzt als value <valuestr>. Wird <keystr> nicht gefunden, so wird der Eintrag an der Stelle erzeugt an der der Cursor sitzt.

10.5.3 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich `$Flag_winst_patch_text_file$ = "on"`

10.6 LinkFolder-Sektionen [W/L]

Mit LinkFolder-Sektionen werden u.a. die Einträge im Startmenü, die Links auf dem Desktop u.ä. verwaltet.

10.6.1 LinkFolder-Sektionen in Windows

Zum Beispiel erzeugt folgende Sektion einen Folder (Ordner) namens *acrobat* im Programme-Folder des allgemeinen Startmenüs (für alle Nutzer gemeinsam).

```
[LinkFolder_Acrobat]
set_basefolder common_programs

set_subfolder "acrobat"
set_link
  name: Acrobat Reader
  target: C:\Programme\adobe\Acrobat\reader\acrord32.exe
  parameters:
  working_dir: C:\Programme\adobe\Acrobat\reader
  icon_file:
  icon_index:
  shortcut:
end_link
```

In einer LinkFolder-Sektion muss zuerst bestimmt werden, in welchem virtuellen Systemfolder die nachfolgenden Anweisungen arbeiten sollen. Dafür existiert die Anweisung

set_basefolder <virtueller Systemfolder>

Virtuelle Windows Systemfolder, die angesprochen werden können, sind:

desktop, sendto, startmenu, startup, programs, desktopdirectory, common_startmenu, common_programs, common_startup, common_desktopdirectory

Die Folder sind virtuell, weil erst durch das Betriebssystem(-Version) bestimmt wird, an welchem physikalischen Ort des Dateisystems sie real existieren.

Im Rahmen einer normalen *Maschinen* Installation sind nur die **common*** Systemfolder relevant.

Die Windows-Systemfolder *desktop, sendto, startmenu, startup, programs, desktopdirectory* können nur im Kontext eines eingeloggten users bzw. in einem *userLoginScript* im Rahmen der opsi-Erweiterung *user Profile Management* verwendet werden.

Im zweiten Schritt werden die Subfolder (bzw. Subfolder-Pfade), in denen Links angelegt werden, mit der Anweisung

set_subfolder <Folderpath>

bestimmt und zugleich geöffnet. Der Subfolder versteht sich absolut (mit Wurzel im gesetzten virtuellen Systemfolder). Wenn direkt im Systemfolder gearbeitet werden soll, wird dieser mit

set_subfolder "" geöffnet.

Im dritten Schritt können die Links gesetzt werden. Der Befehl verwendet eine mehrzeilige Parameterliste. Sie startet mit

set_link

Abgeschlossen wird sie durch

end_link.

Die Parameterliste insgesamt hat folgendes Format:

```
set_link
name: [Linkname]
target: <Pfad und Name des Programms>
parameters: [Aufrufparameter des Programms]
working_dir: [Arbeitsverzeichnis für das Programm]
icon_file: [Pfad und Name der Icon-Datei]
icon_index: [Position des gewünschten Icons in der Icon-Datei]
shortcut: [Tastatur Shortcut zur Aufruf des Programms]
end_link
```

Die Angabe eines *target* ist erforderlich. Alle andere Einträge haben Defaultwerte und können leer sein oder entfallen:

- *name* hat als Defaultwert den Programmnamen,
- *parameters* ist, wenn nichts anderes angegeben ist, ein Leerstring,
- *icon_file* ist, wenn nichts anderes angegeben ist, *target* und
- *icon_index* ist per Default 0.
- *shortcut* ist per Default leer. // since 4.11.6.7
Als *shortcut* darf eine Kombination sein aus [*shift,alt,ctrl*] (nicht case sensitiv) getrennt durch " " (Leerzeichen), "-" (Minuszeichen), "+" (Pluszeichen) sowie einem *Key* oder einem *Virtual Key Code*.
Der *Key* ist ein Buchstabe (*A - Z*) oder eine Zahl (*0 - 9*). Alle anderen Tasten müssen mit Ihrem *Virtual Key Code* Bezeichner eingegeben werden. Diesen erhalten Sie am sichersten über folgendes Hilfsprogramm:
<http://download.uib.de/opsi4.0/helper/showkeys.exe>
Der *shortcut* bezieht sich auf die Tasten und nicht auf deren landesspezifische Belegung. Die Taste *VK_OEM_3* ist in einer deutschen Belegung ein *Ö*, bei einer englischen Belegung ;.
Beispiele für erlaubte shortcuts:

- *O* (Die Taste *O*)
- *VK_O* (Die Taste *O*)
- *Ctrl-O* (Die Kombination *Ctrl O*)
- *Ctrl-Alt-Shift-O* (Die Kombination *Ctrl Alt Shift O*)
- *Ctrl+Alt+Shift+O* (Die Kombination *Ctrl Alt Shift O*)
- *Ctrl Alt Shift O* (Die Kombination *Ctrl Alt Shift O*)
- *Ctrl-Alt-Shift-VK_O* (Die Kombination *Ctrl Alt Shift O*)
- *Ctrl-Alt-Shift-VK_F12* (Die Kombination *Ctrl Alt Shift F12*)

Achtung



Windows: Wenn das referenzierte *target* auf einem, zum Zeitpunkt der Befehlsausführung nicht erreichbaren, Share liegt, werden alle Bestandteile des Pfades auf das Längenschema 8.3 gekürzt.

Workaround:

Manuelles Erzeugen einer korrekten Verknüpfung zu einem Zeitpunkt, in dem das Laufwerk verbunden ist.

Kopieren der korrekten Link-Datei an einen zur Skriptlaufzeit existenten Ort, z.B. *C:\Programme*.

Diese Datei ist dann das Link-*target*.

- `delete_element <Linkname>`
wird der angesprochene Link aus dem geöffneten Folder gelöscht.
- `delete_subfolder <Folderpath>`
löscht den bezeichneten Folder, wobei *Folderpath* als absolut bezüglich des gesetzten virtuellen Systemfolders zu verstehen ist.

10.6.2 Beispiele

```
set $list2$ = createStringList ('common_startmenu', 'common_programs', 'common_startup', '
  common_desktopdirectory')
for $var$ in $list2$ do LinkFolder_Dummy

[LinkFolder_Dummy]
set_basefolder $var$
set_subfolder "Dummy"
set_link
  name: Dummy
```

```

target: C:\Programme\PuTTY\putty.exe
parameters:
working_dir: C:\Programme\PuTTY
icon_file:
icon_index:
end_link

```

Ergibt folgenden Log:

```

Set $list2$ = createStringList ('common_startmenu', 'common_programs', 'common_startup', '
common_desktopdirectory')
retrieving strings from createStringList [switch to loglevel 7 for debugging]
(string 0)common_startmenu
(string 1)common_programs
(string 2)common_startup
(string 3)common_desktopdirectory

retrieving strings from $list2$ [switch to loglevel 7 for debugging]
(string 0)common_startmenu
(string 1)common_programs
(string 2)common_startup
(string 3)common_desktopdirectory

~~~~~ Looping through: 'common_startmenu', 'common_programs', 'common_startup', '
common_desktopdirectory'

Execution of LinkFolder_Dummy
Base folder is the COMMON STARTMENU folder
Created "Dummy" in the COMMON STARTMENU folder
ShellLink "Dummy" created

Execution of LinkFolder_Dummy
Base folder is the COMMON PROGRAMS folder
Created "Dummy" in the COMMON PROGRAMS folder
ShellLink "Dummy" created

Execution of LinkFolder_Dummy
Base folder is the COMMON STARTUP folder
Created "Dummy" in the COMMON STARTUP folder
ShellLink "Dummy" created

Execution of LinkFolder_Dummy
Base folder is the COMMON DESKTOPDIRECTORY folder
Created "Dummy" in the COMMON DESKTOPDIRECTORY folder
ShellLink "Dummy" created

~~~~~ End Loop

```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich `$Flag_winst_link_folder$ = "on"`.

10.6.3 LinkFolder-Sektionen in Linux

LinkFolder Sektionen werden jetzt auch unter Linux unterstützt.

Erlaubte BaseFolder sind: `common_programs,common_autostart,desktop, autostart`

Subfolder ist immer "" (leer). Die Link Option `icon_index` wird ignoriert.

Als zusätzliche Link Option gibt es: `link_categories`. Hier sind folgende durch Semikolon getrennt und abgeschlossene Werte erlaubt: `AudioVideo`, `Audio`, `Video`, `Development`, `Education`, `Game`, `Graphics`, `Network`, `Office`, `Settings`, `System`, `Utility`

Die LinkFolder Sektion unter Linux funktioniert für unterschiedliche Desktopsysteme.

10.7 XMLPatch-Sektionen [W]

Immer häufiger werden Daten aller Art, insbesondere auch Konfigurationsdaten, als XML-Dokument gespeichert.

Der *opsi-winst/opsi-script* bietet XMLPatch-Sektionen an, um XML-Dokumente zu bearbeiten.

Ähnlich wie bei anderen Sektionen (Registry, Patches, LinkFolder) wird dazu zunächst mit bestimmten Befehlen an die Stelle navigiert, an der gearbeitet werden soll und dann dort Detailkommandos ausgeführt.

Das bedeutet, die Aktionen, die *opsi-winst/opsi-script* ausführen kann, gliedern sich in:

- **die Selektion** eines Sets von Elementen des XML-Dokuments, inklusive der Erzeugung nicht vorhandener Elemente,
- **Patch-Aktionen**, die für alle Elemente eines Sets ausgeführt werden sowie
- **die Ausgabe** von Namen und/Attributen der selektierten Elemente für die weitere Verarbeitung.

10.7.1 Aufrufparameter

Der Name der zu patchenden Datei wird als Parameter übergeben.

Beispiel:

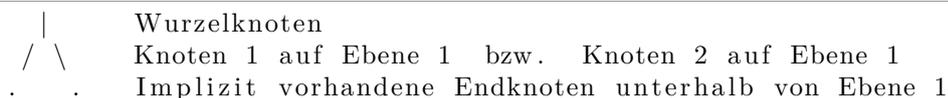
```
XMLPatch_mozilla_mimetypes $mozillaprofilepath$ + "\mimetypes.rdf"
```

10.7.2 Struktur eines XML-Dokuments

Ein XML-Dokument beschreibt die Logik eines „Baums“ (tree), der sich ausgehend von einer „Wurzel“ (root) – passenderweise document root genannt – in die „Äste“ (branches) verzweigt. Jede Verzweigungsstelle, wie auch jedes „Astende“, wird als „Knoten“ bezeichnet (englisch node). Die nachgeordneten Knoten eines Knotens heißen auch Kinderknoten ihres Elternknotens.

In XML wird dieser Baum konstruiert durch Elemente. Der Anfang der Beschreibung eines Elements ist mit einem Tag gekennzeichnet (ähnlich wie in der Web-Auszeichnungssprache HTML), d.h. durch einen spezifischen Markierungstext, der durch „<“ und „>“ umrahmt ist. Das Ende der Beschreibung wird wieder durch ein Tag desselben Typnamens gekennzeichnet, jetzt aber durch „</“ und „>“ geklammert. Wenn es keine nachgeordneten Elemente gibt, kann die getrennte Endmarkierung entfallen, stattdessen wird das öffnende Tag mit „/>“ abgeschlossen.

Einen „V“-Baum – mit einer einzigen Verzweigung in zwei Teiläste – könnte man so skizzieren (Wurzel nach oben gedreht):



Er würde in XML folgendermaßen dargestellt:

```
<?xml version=" 1.0 " ?>
<Wurzelknoten>
  <Knoten_Ebene-1_Nummer-1>
</Knoten_Ebene-1_Nummer-1>
  <Knoten_Ebene-1_Nummer-2>
</Knoten_Ebene-1_Nummer-2>
</Wurzelknoten>
```

Die erste Zeile benennt nur die XML-Definition nach der sich das Dokument richtet. Die weiteren Zeilen beschreiben den Baum.

Die insoweit noch nicht komplizierte Struktur wird dadurch verwickelt, dass bis jetzt nur „Hauptknoten“ vorkommen. Ein Hauptknoten definiert ein „Element“ des Baums und ist durch ein Tag gekennzeichnet. Einem solchen Hauptknoten können – wie bei der Skizze schon angedeutet – „Unterknoten“ und sogar mehrere Arten davon zugeordnet sein. (Befände sich der Baum in der normalen Lage mit Wurzel nach unten, müssten die Unterknoten „Überknoten“ heißen.) Folgende Arten von Unterknoten sind zu berücksichtigen:

- Nachgeordnete Elemente, z.B. könnte der Knoten Nummer 1 sich in Subknoten A bis C verzweigen:

```
<Knoten_Ebene-1_Nummer-1>
  <Knoten_Ebene-2_A>
</Knoten_Ebene-2_A>
  <Knoten_Ebene-2_B>
</Knoten_Ebene-2_B>
  <Knoten_Ebene-2_C>
</Knoten_Ebene-2_c>
</Knoten_Ebene-1_Nummer-1>
```

- Nur wenn es KEINE nachgeordneten Elemente gibt, kann das Element Text enthalten. Dann heißt es, dass dem Element ein Textknoten untergeordnet ist. Beispiel:

```
<Knoten_Ebene-1_Nummer-2>Hallo Welt
</Knoten_Ebene-1_Nummer-2>
```

- Der Zeilenumbruch, der zuvor nur Darstellungsmittel für die XML-Struktur war, zählt dabei jetzt auch als Teil des Textes! Wenn er nicht vorhanden sein soll, muss geschrieben werden

```
<Knoten_Ebene-1_Nummer-2>Hallo Welt</Knoten_Ebene-1_Nummer-2>
```

- Zum Element können außer dem Hauptknoten noch Attribute, sog. Attributknoten gehören. Es könnte z.B. Attribute „Farbe“ oder „Winkel“ geben, die den Knoten 1 in der Ebene 1 näher beschreiben.

```
<Knoten_Ebene-1_Nummer-1 Farbe="grün" Winkel="65">
</Knoten_Ebene-1_Nummer-1>
```

Eine derartige nähere Beschreibung eines Elements ist mit beiden anderen Arten von Unterknoten vereinbar.

Zur Auswahl einer bestimmten Menge von Elemente könnten im Prinzip alle denkbaren Informationen herangezogen werden, insbesondere

1. die Elementebene (Schachtelungstiefe im Baum),
2. der Name der Elemente, d.h. Name der entsprechenden Hauptknoten, in der Abfolge der durchlaufenen Ebenen (der „XML-Pfad“),
3. die Anzahl, Namen und Werte der zusätzlich gesetzten Attribute,
4. die Reihenfolge der Attribute,
5. die Reihenfolge der Elemente,
6. sonstige „Verwandtschaftsbeziehungen“ der Elemente und
7. Text-(Knoten-)Inhalte von Elementen.

Im *opsi-winst/opsi-script* ist derzeit die Auswahl nach den Gesichtspunkten (1) bis (3) sowie (7) implementiert:

10.7.3 Optionen zur Bestimmung eines Sets von Elementen

Vor jeder weiteren Operation muss das Set von Elementen bzw. von Hauptknoten bestimmt werden, auf die sich die Operation beziehen soll. Das Set wird Schritt für Schritt ermittelt, indem ausgehend von der Dokumentenwurzel Pfade gebildet werden, die jeweils über akzeptierte nachgeordnete Elemente laufen. Die letzten Elemente der Pfade bilden dann das ausgewählte Set.

Der *opsi-winst/opsi-script* Befehl hierfür lautet

- `OpenNodeSet`

Für die Festlegung der akzeptierten Pfade existiert eine ausführliche und eine Kurzsyntax.

Ausführliche Syntax Die ausführliche Syntax für die Beschreibung eines Elemente-Sets bzw. einer Knoten-Menge ist in der folgenden Variante eines Beispiels zu sehen (vgl. Kochbuch, [Kapitel "XML-Datei patchen"](#)):

```
openNodeSet
  documentroot
  all_childelements_with:
    elementname:"define"
  all_childelements_with:
    elementname:"handler"
    attribute: extension value="doc"
  all_childelements_with:
    elementname:"application"
end
```

Kurzsyntax Das gleiche Nodeset beschreibt folgende Kurzsyntax (muss in einer Zeile des Skripts untergebracht werden):

```
openNodeSet 'define /handler value="doc"/application /'
```

In dieser Syntax separieren die Schrägstriche die Schritte innerhalb der Baumstruktur, welche in einer Syntax angegeben werden, die ausführlicher als eine eigene Beschreibung ist.

Selektion nach Text-Inhalten (nur ausführliche Syntax) Die ausführliche Syntax erlaubt auch die Selektion nach Text-Inhalten eines Tags:

```
openNodeSet

  documentroot
  all_childelements_with:
  all_childelements_with:
    elementname:"description"
    attribute:"type" value="browser"
    attribute:"name" value="mozilla"
  all_childelements_with:
    elementname:"linkurl"
    text:"http://www.mozilla.org"
end
```

Parametrisierung der Suchstrategie Bei den bislang aufgeführten Beschreibungen eines Elemente-Sets bleiben allerdings eine ganze Reihe von Fragen offen.

- Soll ein Element akzeptiert werden, wenn der Elementname und die aufgeführten Attribute passen, aber weitere Attribute existieren?
- Soll die Beschreibung im Ergebnis eindeutig sein, d.h. genau ein Element liefern? Und wenn doch die Beschreibung des Pfades auf mehrere Elemente passt, muss dann möglicherweise von einer nicht korrekten Konfigurationsdatei ausgegangen werden?

- Soll umgekehrt auf jeden Fall ein passendes Element erzeugt werden, wenn keines existiert?

Zur Regelung dieser Fragen kann die OpenNodeSet-Anweisung parametrisiert werden. Bei den nachfolgend genannten Parametern überdecken „stärkere“ Einstellungen „schwächere“, z.B. ersetzt eine Fehlermeldung eine ansonsten geforderte Warnung. Die angegebenen booleschen Werte sind die Default-Werte:

```
- error_when_no_node_existing false
- warning_when_no_node_existing true
- error_when_nodccount_greater_1 false
- warning_when_nodccount_greater_1 false
- create_when_node_not_existing false
- attributes_strict false
```

Bei Verwendung der Kurzsyntax der OpenNodeSet-Anweisung muss die Parametrisierung vorausgehen und gilt für alle Ebenen des XML-Baumes. In der ausführlichen Syntax kann sie auch direkt nach der OpenNodeSet-Anweisung erfolgen oder für jede Ebene neu gesetzt werden. Sinnvoll kann letzteres vor allem für die Einstellung der Option „create when node not existing“ (Erstellung von Knoten, wenn es keine gibt) sein.

10.7.4 Patch-Aktionen

Auf der mit OpenNodeSet geöffneten bzw. erzeugten Knotenmenge arbeiten nachfolgende Patch-Anweisungen. Es existieren solche:

- zum Setzen und Löschen von Attributen,
- zum Entfernen von Elementen und
- zum Setzen von Text.
- **SetAttribute** "Attributname" value="Attributwert"
setzt in jedem Element des aktuellen Knoten- bzw. Elementsets das Attribut auf den genannten Wert. Wenn das Attribut nicht vorhanden ist wird es erzeugt.
Beispiel:
SetAttribute "name" value="OpenOffice Writer"
- **AddAttribute** "Attributname" value="Attributwert"
setzt das Attribut dagegen nur auf Attributwert, wenn es vorher nicht existiert, ein vorhandenes Attribut behält seinen Wert. Z.B. würde die Anweisung
AddAttribute "name" value="OpenOffice Writer"
eine vorher vorhandene Festlegung auf ein anderes Programm nicht überschreiben.
- **DeleteAttribute** "Attributname"
wird das betreffende Attribut von jedem Element der aktuellen Knotenmenge entfernt.
- **DeleteElement** "Elementname"
entfernt das Element, dessen Hauptknoten den (Tag-) Namen "Elementname" hat, samt Unterknoten aus der aktuellen Knoten- oder Elementmenge.

Schließlich existieren zwei Anweisungen zum Setzen bzw. Hinzufügen von Textinhalten eines Elements. Die beiden Anweisungen lauten

- **SetText** "Text"

und

- **AddText** "Text"

Z.B. wird, wenn das betreffende Element in der geöffneten Elementmenge liegt, durch die Anweisung

```
SetText "rtf"
```

aus

```
<fileExtensions>doc<fileExtensions>
```

das Element

```
<fileExtensions>rtf<fileExtensions>
```

Mit

```
SetText ""
```

wird der Text komplett entfernt.

```
AddText "rtf"
```

setzt analog wie bei anderen Add-Anweisungen den Text, sofern kein Text vorhanden ist - existierender Text bleibt unberührt.

10.7.5 Rückgaben an das aufrufende Programm

Eine XMLPatch-Sektion kann angewiesen werden, String-Listen an das rufende Programm zurückzugeben.

Dazu muss sie in einer primären Sektion mit der String-Listen-Anweisung `getReturnListFromSection` aufgerufen werden. Die Anweisung kann in einem String-Listen-Ausdruck verwendet werden, z.B. das Ergebnis einer String-Listen-Variable zugewiesen werden. So kann in der XMLPatch_mime-Sektion stehen:

```
DefStringList $list1$
set $list1$=getReturnListFromSection ('XMLPatch_mime "c:\mimetypes.rdf"')
```

Eine Return-Anweisung in der XMLPatch-Sektion regelt, welche Zeilen die XMLPatch-Sektion als Inhalt der String-Liste ermittelt:

- `return elements+` Bewirkt, dass die ausgewählten Elemente komplett (Elementname und Attribute) ausgegeben werden.
- `return attributes`
Erzeugt eine Liste der Attribute.
- `return elementnames`
Listet die Elementnamen.
- `return attributenames` Produziert eine Liste der Attributnamen.
- `return text`
Listet die textlichen Inhalte der selektierten Elemente.
- `return counting`
Liefert eine Listenstruktur mit summarischen Informationen: In Zeile 0 steht die Anzahl aller ausgewählten Elemente, in Zeile 1 die Zahl aller Attribute.

10.7.6 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich `$Flag_winst_xml$ = "on"`

10.8 ProgmanGroups-Sektionen

Dieser Sektionstyp ist abgekündigt.

10.9 WinBatch-Sektionen [W/L]

In einer WinBatch-Sektion kann jedes Windows-Programm als Anweisung verwendet werden.

Z.B kann mit folgender WinBatch-Sektion ein Setup-Programm gestartet werden:

```
[winbatch_install]
"%scriptpath%\setup.exe"
```

Winbatch Sektionen dienen dazu **Programme** (*.exe) aufzurufen.

Der Aufruf von anderen Dateien, die mit einem Programm verknüpft sind, direkt aufzurufen ist abgekündigt (aber noch unterstützt). Wenn Sie das tun bekommen Sie eine deprecated Warnung. Beispiel:

ok: notepad.exe test.txt

depricated (not ok): test.txt

10.9.1 Aufrufparameter (Modifier)

Durch die Parameter des WinBatch-Aufrufs wird festgelegt, wie sich *opsi-winst/opsi-script* gegenüber den in der WinBatch-Sektion gestarteten Programmen verhält.

- `/32Bit` //seit 4.11.3.5 [W]
Das ist der Default. Die in der Sektion angegebene Pfade werden als 32 Bit Pfade interpretiert.
Beispiel: `c:\windows\system32\regedit.exe` ruft (auch auf einem 64bit System) die 32 Bit *regedit.exe* auf.
- `/64Bit` //seit 4.11.3.5 [W]
Die in der Sektion angegebene Pfade werden als 64 Bit Pfade interpretiert.
Beispiel: `c:\windows\system32\regedit.exe` ruft (auf einem 64bit System) die 64 Bit *regedit.exe* auf.
- `/SysNative` //seit 4.11.3.5 [W]
Die in der Sektion angegebene Pfade werden gemäß der OS Architektur interpretiert.
Beispiel: `c:\windows\system32\regedit.exe` ruft auf einem 64bit System die 64 Bit *regedit.exe* und auf einem 32bit System die 32 Bit 'regedit.exe' auf.

Beispiel:

```
Winbatch_add_reg /64Bit
[Winbatch_add_reg]
"c:\windows\system32\regedit.exe" /s "%scriptpath%\my64.reg"
```

- `/WaitOnClose`
Default
opsi-winst wartet die Selbstbeendigung des angestoßenen Prozesses ab. Dieses Verhalten kann mit dem Parameter auch explizit definiert werden.
- `/LetThemGo`
Verschiebt den aufgerufenen Prozess in den Hintergrund und wartet **nicht** auf dessen Beendigung; d.h. das sofort die nächste Zeile der WinBatch-Sektion bzw. die nächste Zeile des übergeordneten Programms abgearbeitet werden.
- `/WaitSeconds` [AnzahlSekunden]
Die Parametrisierung `/WaitSeconds` [AnzahlSekunden] modifiziert das Verhalten dahingehend, dass *opsi-winst/opsi-script* jeweils erst nach [AnzahlSekunden] die Skriptbearbeitung fortsetzt. Die angegebene Zeit stoppt *opsi-winst/opsi-script* auf jeden Fall. In der Default-Einstellung wird zusätzlich auf das Ende der angestoßenen Prozesse gewartet. Ist letzteres nicht gewünscht, so kann der Parameter mit dem Parameter `/LetThemGo` kombiniert werden.

- `/WaitForWindowAppearing [Fenstertitel] [W]`
bzw.

`/WaitForWindowVanish [Fenstertitel] [W]`

Abgekündigt. Verwenden Sie `/WaitForProcessEnding`

Im 1. Fall wartet *opsi-winst/opsi-script* solange, bis ein Prozess, der sich durch ein mit [Fenstertitel] benanntes Fenster kenntlich macht, gestartet ist. Im 2. Fall wartet *opsi-winst/opsi-script* bis ein, mit [Fenstertitel] benanntes, Fenster auf dem Desktop erst einmal erscheint und dann auch wieder geschlossen wird. Auf diese Weise kann unter geeigneten Umständen geprüft werden, ob sekundäre, indirekt gestartete Prozesse sich beendet haben.



Achtung

Diese Befehle erkennen nur Fenster von 32 Bit-Programmen.

- `/WaitForProcessEnding <program name>`

Wartet darauf, das sich der Prozess mit dem Namen <program name> beendet.

Kann und sollte mit `/TimeoutSeconds` kombiniert werden.

Erläuterung:

Der *opsi-winst/opsi-script* wartet auf das Ende eines per `winbatch` gestarteten Prozesses bevor mit der nächsten Zeile des Scriptes vorgefahren wird:

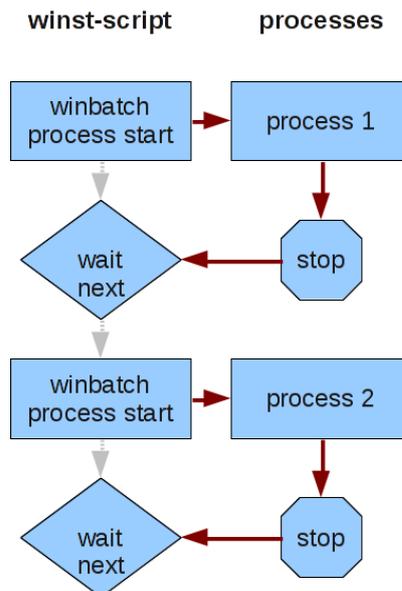


Abbildung 10.1: Sequentielle Abarbeitung des Scriptes mit Warten auf das Ende eines Prozesses

Es gibt allerdings Prozesse, welche einen weiteren Prozess starten und sich Beenden ohne auf das Ende des Kindprozesses zu warten. Aus Sicht des *opsi-winst/opsi-script* ist damit der Weg zur Ausführung des nächsten Befehls frei:

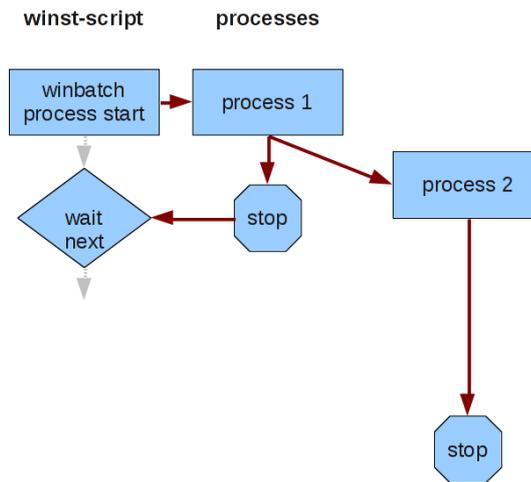


Abbildung 10.2: Ende eines Prozesses mit weiterlaufenden Kindprozess

Werden z.B. hintereinander ein Uninstall und ein Setup Programm aufgerufen und das Uninstall Programm führt die eigentliche Deinstallation in einem Kindprozess aus, so ist das Ergebnis undefiniert, da deinstallation und Installation gleichzeitig laufen:

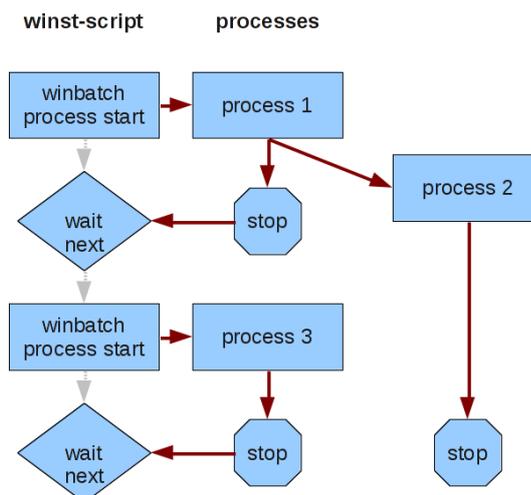


Abbildung 10.3: Überlappung von Kindprozess und nächstem gestarteten Prozess

Mit dem Modifier `/WaitForProcessEnding` kann eine solche Situation vermieden werden.

- `/TimeoutSeconds <seconds>`
Bricht das Warten auf das Prozessende oder eine Wartebedingung (`/WaitForProcessEnding`) nach Ablauf von `<seconds>` ab, auch wenn das Prozessende oder die Wartebedingung noch nicht erfüllt ist. Der Prozess auf dessen Ende gewartet werden sollte wird nicht gestopt. Kann seit Version 4.11.3 auch alleine (z.B. ohne `/WaitForProcessEnding`) verwendet werden, aber nicht zusammen mit `/WaitSeconds`. Seit 4.11.4.6 wird der Zeitablauf bis zum Timeout über den Fortschrittsbalken angegeben. Beispiel:

```
Winbatch_uninstall /WaitForProcessEnding "uninstall.exe" /TimeoutSeconds 20
[Winbatch_uninstall]
"%ScriptPath%\uninstall_starter.exe"
```

- `/RunElevated` [W]
Started dem Prozeß mit einem höheren Security Token (d.h. mit höheren Rechten). Dieser Modifier hat folgende Einschränkungen:
 - Unter NT5 hat er keine Auswirkungen
 - Ein Zugriff auf das Netz ist in dem Prozess nicht möglich. Daher müssen die aufzurufenden Programme von einem Netzlaufwerk in ein temporäres lokales Verzeichnis kopiert werden.
 - Evtl. kann es zu Problemen bei der Nutzung der grafischen Oberfläche kommen. Daher sind echte silent aufrufe hier zu bevorzugen.
 - Funktioniert nur im opsi-service Kontext
- `/RunAsLoggedonUser` //seit 4.11.3.5 [W]
Nur im Kontext eines *userLoginScripts* verfügbar. Startet das Programm als der user der sich gerade einlogt. Dieser Modifier hat folgende Einschränkungen:
 - Unter NT6 wenig getestet und evtl. nur Eingeschränkt wirksam.
- `getLastExitCode`
Die String-Funktion `getLastExitCode` gibt den ExitCode des letzten Prozessaufrufs der vorausgehenden WinBatch / DosBatch / ExecWith Sektion aus.

10.9.2 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-script-test* und dort den Bereich `$Flag_winst_winbatch$ = "on"`

10.10 DOSBatch/DosInAnIcon (ShellBatch/ShellInAnIcon) Sektionen [W/L]

DOSBatch-Sektionen (auch ShellBatch genannt) sollen in erster Linie dazu dienen, vorhandene Kommandozeilenroutinen für bestimmte Zwecke zu nutzen. *opsi-winst/opsi-script* wartet auf die Beendigung des DOS-Batch, bevor die nächste Sektion des Skripts abgearbeitet wird.

Eine DosBatch-Sektion wird bei der Abarbeitung des Skripts in eine temporäre Batch-Datei `winst<random>.cmd` umgewandelt. Da die Datei in `c:\opsi.org\tmp` angelegt wird, muss dieses Verzeichnis existieren und zugänglich sein. Die Batch-Datei wird dann in einem Kommando-Fenster mit `cmd.exe` als Kommando-Interpreter ausgeführt. Dies erklärt warum in einer DosBatch Sektion alle Windows Shell Kommandos verwendet werden können.

Gegenüber dem Aufruf einer `cmd`-Datei per Winbatch-Sektion bietet die DosBatch Sektionen drei Vorteile:

- In der Sektion vorhandene *opsi-winst/opsi-script* Variablen oder Konstanten werden vor der Ausführung durch Ihren Inhalt ersetzt und können so unkompliziert verwendet werden.
- Die Ausgaben eines `DosInAnIcon/ShellInAnIcon` Aufrufs unter Windows werden in der Logdatei abgespeichert.
- Die Ausgaben eines `DosInAnIcon/ShellInAnIcon` Aufrufs werden in einem gesonderten Fenster ausgegeben, enn die Sektion mit dem Parameter `/showoutput` aufgerufen wurde.
- Die Ausgaben eines `DosInAnIcon/ShellInAnIcon` bzw. `DosBatch/ShellBatch` Aufrufs unter Linux werden in der Logdatei abgespeichert.
- Die Ausgaben des Aufrufs können einer String-Liste übergeben und weiterverarbeitet werden.

Der Sektionstyp `DOSInAnIcon` oder `ShellInAnIcon` ist identisch mit der betreffenden `DOSBatch/ShellBatch` Syntax und ausführenden Methoden. Allerdings wird das aufgerufenen Fenster minimiert dargestellt und die Ausgaben in die Logdatei übernommen.

**Achtung**

Verwenden Sie keine Kommandos, die auf Eingaben warten.

10.10.1 Aufrufparameter

Zu unterscheiden ist zwischen Parametern die der aufgerufenen Batchdatei übergeben werden und denen die *opsi-winst/opsi-script*-intern verwendet werden. Der Aufrufsyntax ist daher:

Sektionsname [batch parameter] [winst [modifier]]

Erlaubte winst modifier sind (seit 4.11.1):

- /32bit
- /64bit
- /Sysnative
- /showoutput // since 4.11.4.6

Parameter des Aufrufs der DosBatch-Sektion in der Actions-Sektion werden unmittelbar als Parameter der Batch-Datei interpretiert.

Zum Beispiel bewirken die Anweisungen in Actions-Sektionen bzw. der Sektion DosBatch_1 :

```
[Actions]
DefVar $para$
DosBatch_1 today we say "Hello World"
set $para$ = "today"
DosBatch_1 $para$ we say "Hello World"

[DosBatch_1]
@echo off
echo %1 %2 %3 %4
pause
```

die Ausführung des Dos-Batch-Befehls echo mit Parametern *today we say "Hello World"*.

Das folgende Beispiel wird auf einem 64 Bit System mit einer 64 Bit cmd.exe gestartet und erzeugt die Ausgabe *today we say*:

```
[Actions]
DosBatch_1 today we say winst /64bit

[DosBatch_1]
@echo off
echo %1 %2 %3 %4
pause
```

Seit Version 4.11.5 sind als Parameter neben Stringkonstanten auch Stringvariablen erlaubt (aber keine String Funktionen).

Example:

Code from opsi-script-test:

```

comment "Testing parameters for ShellBatch"
set $ConstTest$ = "Hello world"
set $list$ = getOutputStreamFromSection('DosInAnIcon_with_parameter world')
set $CompValue$ = takeString(2,$list$)
if ($ConstTest$ = $CompValue$)
    comment "ShellBatch parameter passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "ShellBatch parameter failed"
endif

comment "Testing parameters for ShellBatch"
set $ConstTest$ = "Hello world"
set $tmp$ = "world"
set $list$ = getOutputStreamFromSection('DosInAnIcon_with_parameter $tmp$')
set $CompValue$ = takeString(2,$list$)
if ($ConstTest$ = $CompValue$)
    comment "ShellBatch parameter passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "ShellBatch parameter failed"
endif

```

ergibt den Log:

```

comment: Testing parameters for ShellBatch
Set $ConstTest$ = "Hello world"
    The value of the variable "$ConstTest$" is now: "Hello world"
Set $list$ = getOutputStreamFromSection('DosInAnIcon_with_parameter world')

DosInAnIcon_with_parameter
c:\opsi.org\tmp\_opsiscript_Kj23Ej02.cmd saved back
Executing "cmd.exe" /C c:\opsi.org\tmp\_opsiscript_Kj23Ej02.cmd world
ExitCode 0

    output:
    -----

    C:\Windows\system32>echo Hello world
    Hello world

The file: c:\opsi.org\tmp\_opsiscript_Kj23Ej02.cmd has been deleted
retrieving strings from getOutputStreamFromSection [switch to loglevel 7 for debugging]
(string 0)
(string 1)C:\Windows\system32>echo Hello world
(string 2)Hello world

Set $CompValue$ = takeString(2,$list$)
retrieving strings from $list$ [switch to loglevel 7 for debugging]
(string 0)
(string 1)C:\Windows\system32>echo Hello world
(string 2)Hello world

The value of the variable "$CompValue$" is now: "Hello world"
If
$ConstTest$ = $CompValue$ <<< result true
($ConstTest$ = $CompValue$) <<< result true

```

```

Then
  comment: ShellBatch parameter passed
Else
EndIf

comment: Testing parameters for ShellBatch
Set $ConstTest$ = "Hello world"
  The value of the variable "$ConstTest$" is now: "Hello world"
Set $tmp$ = "world"
  The value of the variable "$tmp$" is now: "world"
Set $list$ = getOutputStreamFromSection('DosInAnIcon_with_parameter $tmp$')

DosInAnIcon_with_parameter
c:\opsi.org\tmp\_opsiscript_Kz50Gi50.cmd saved back
Executing "cmd.exe" /C c:\opsi.org\tmp\_opsiscript_Kz50Gi50.cmd world
ExitCode 0

      output:
      -----

      C:\Windows\system32>echo Hello world
      Hello world

The file: c:\opsi.org\tmp\_opsiscript_Kz50Gi50.cmd has been deleted
retrieving strings from getOutputStreamFromSection [switch to loglevel 7 for debugging]
(string 0)
(string 1)C:\Windows\system32>echo Hello world
(string 2)Hello world

Set $CompValue$ = takeString(2,$list$)
  retrieving strings from $list$ [switch to loglevel 7 for debugging]
  (string 0)
  (string 1)C:\Windows\system32>echo Hello world
  (string 2)Hello world

The value of the variable "$CompValue$" is now: "Hello world"
If
  $ConstTest$ = $CompValue$ <<< result true
  ($ConstTest$ = $CompValue$) <<< result true
Then
  comment: ShellBatch parameter passed
Else
EndIf

```

10.10.2 Einfangen der Ausgaben

Sollen die Ausgaben, die von Befehlen einer DosBatch-Sektion kommen, aufgefangen werden, so geschieht dies mittels `getOutputStreamFromSection ()` aus der Haupt-Sektion des opsi-winst-Skripts (siehe [Kapitel "\(Wieder-\) Gewinnen von Einzelstrings aus String-Listen"](#)).

Sollen die zurückgegebenen Strings weiterverarbeitet werden, so wird dringend geraten, vor den Befehlszeilen ein `@`-Zeichen zu verwenden bzw. die Kommandos mit `@echo off` zu beginnen. Dies unterdrückt die Ausgabe der Befehlszeile selbst, die je nach System anders formatiert sein kann.

10.10.3 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-script-test* und dort den Bereich $\$Flag_winst_dos\$ = "on"$

10.11 Registry-Sektionen [W]

Diese Funktion ist nur unter Windows verfügbar.

Registry-Sektionen dienen dem Erzeugen und Patchen von Einträgen in der Windows-Registrierdatenbank, wobei die Eintragungen mit dem opsi-winst-üblichen Detaillierungsgrad protokolliert werden.

10.11.1 Beispiele

Man kann eine Registry-Variable setzen indem man die Sektion mit `Registry_TestPatch` aufruft, wo sie dann wie folgt angegeben ist

```
[Registry_TestPatch]
openkey [HKEY_Current_User\Environment\Test]
set "Testvar1" = "c:\rutils;%Systemroot%\hey"
set "Testvar2" = REG_DWORD:0001
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich $\$Flag_subregistry\$ = "on"$

10.11.2 Aufrufparameter

- Die Standardform der Registry-Sektionen ist unparametrisiert. Dies genügt, weil auf dem Windows-PC nur eine einzige Registrierdatenbank gibt und somit das globale Ziel der Bearbeitung feststeht.
- `/AllNTUserDats`
Es gibt jedoch die Möglichkeit, dass die Patches einer Registry-Sektion automatisch für "alle NT User", entsprechend den verschiedenen User-Zweigen der Registry, vorgenommen werden. Das entsprechende Verfahren bei der Abarbeitung der Sektion wird mit dem Parameter `/AllNTUserDats` aufgerufen.

Außerdem kontrollieren Parameter mit welchen syntaktische Varianten Registry-Sektionen angefordert werden kann:

- `/regedit`
Wird das Registry-Kommando mit dem Parameter `/regedit` verwendet, so kann der Export eines Registry-Teilzweiges mit dem Programm, der mit dem gewöhnlichen Windows-Registry-Editor `regedit` erstellt wurde, direkt als Eingabedatei für Registry dienen (vgl. Abschnitt "[Registry-Sektionen im Regedit-Format](#)").
- `/addReg`
Eine weitere Variante des Registry-Aufrufs dient dazu, die Patch-Anweisungen für die Registry zu verarbeiten, die im inf-Datei-Standard erstellt sind. Zur Kennzeichnung dient der Parameter `/addReg` (in Anlehnung an die entsprechende Abschnittsbezeichnung in einer inf-Datei)(vgl. Abschnitt "[Registry-Sektionen im AddReg-Format](#)").

Diese nicht *opsi-winst/opsi-script* spezifischen syntaktischen Varianten sind im Handbuch nicht beschrieben, da sie normalerweise automatisch generiert werden.

Weiterhin gibt es die Aufrufparameter,

- `/32Bit`
- `/64Bit`

- /SysNative

welche auf 64 Bit-Systemen das Schreiben in den 32 Bit- bzw. 64 Bit-Zweig der Registry beeinflusst (siehe [Kapitel 64 Bit-Unterstützung](#)).

10.11.3 Kommandos

Die Syntax der Defaultform einer Registry-Sektion ist an der Kommandosyntax anderer Patchoperationen des *opsi-winst/opsi-script* orientiert.

Es existieren die Anweisungen:

- OpenKey
- Set
- Add
- Supp
- GetMultiSZFromFile
- SaveValueToFile
- DeleteVar
- DeleteKey
- ReconstructFrom
- Flushkey

Im Detail:

- **OpenKey** <Registrieschlüssel>
Öffnet den bezeichneten Schlüssel in der Registry zum Lesen (und wenn der eingeloggte User über die erforderlichen Rechte verfügt zum Schreiben); existiert der Schlüssel noch nicht, wird er erzeugt.

Registry-Schlüssel sind ja hierarchisch organisierte Einträge Registrierungsdatenbank. Die hierarchische Organisation drückt sich in der mehrstufigen Benennung aus: Für die oberste (Root-) Ebene können standardmäßig insbesondere die "high keys" *HKEY_CLASSES_ROOT*, *HKEY_CURRENT_USER*, *HKEY_LOCAL_MACHINE*, *HKEY_USERS* und *HKEY_CURRENT_CONFIG* verwendet werden. Gebräuchliche Abkürzungen sind *HKCR*, *HKCU*, *HKLM* und *HKU*.

In der *opsi-winst/opsi-script* Syntax bei den Registry-Pfaden werden die weiteren folgenden Ebenen jeweils durch einen Backslash getrennt.

Alle anderen Kommandos arbeiten mit einem geöffneten Registry-Key.

- **Set** <Varname> = <Value>
setzt die durch <Varname> bezeichnete Registry-Variable auf den Wert <Value>, wobei es sich sowohl bei <Varname> als auch bei <Value> um Strings handelt, die in Anführungszeichen eingeschlossen sind. Existiert die Variable noch nicht, wird sie erzeugt. Dabei wird als Default Datentyp *REG_SZ* verwendet. Enthält allerdings <value> ein oder mehrere Prozentzeichen (%) so wird als Datentyp *REG_EXPAND_SZ* verwendet.

Es gibt auch den Leerstring als Variablenname; dieser entspricht dem "(Standard)"-Eintrag im Registry-Schlüssel.

Soll eine Registry-Variable erzeugt oder gesetzt werden, bei der der Datentyp explizit angegeben werden soll, muss die erweiterte Form der Set-Anweisung verwendet werden:

- **Set** <Varname> = <Registrytyp>:<Value>
Setzt die durch <Varname> bezeichnete Registry-Variable auf den Wert <Value> des Typs <Registrytyp>. Es werden folgende Registry-Typen interpretiert:

REG_SZ

(String)

REG_EXPAND_SZ

(ein String, der vom System zu expandierende Teilstrings wie %Systemroot% enthält)

REG_DWORD

(ganzzahlige Werte; Dezimaldarstellung oder 0xHexadezimal)

REG_BINARY

(binäre Werte, in zweistelligen Hexadezimalen, d.h. 00 01 02 .. 0F 10 .., notiert)

REG_MULTI_SZ

(Arrays von String-Werten, die in opsi-winst-Syntax durch das Zeichen "|" getrennt werden;

Beispiel für REG_MULTI_SZ:

```
set "myVariable" = REG_MULTI_SZ:"A|BC|de"
```

Wenn ein Multi-String zunächst zusammengestellt werden soll, kann dies zeilenweise in einer Datei geschehen, die man dann mit Hilfe der Anweisung GetMultiSZFromFile (s.u.) einliest.

Beispiel für **set** mit unterschiedlichen Registrydatentypen:

```
set "var1" = "my string"
set "var2" = REG_SZ:"my string"
set "var3" = REG_EXPAND_SZ:"%ProgramFiles%"
set "var4" = REG_DWORD:123      (Decimal)
set "var5" = REG_DWORD:0x7b    (Hexadecimal)
set "var6" = REG_BINARY:00 01 02 0F 10
set "var7" = REG_MULTI_SZ:"A|BC|de"
```

- **Add** <Varname> = <Value>
bzw.
Add <Varname> = <Registrytyp> <Value>
arbeitet analog zu Set mit dem Unterschied, dass nur Variablen hinzugefügt, Einträge für bestehende Variablen nicht verändert werden.
- **Supp** <Varname> <Listenzeichen> <Supplement>
Dieses Kommando liest den String-Wert der Variablen <varname>, einer Liste aus Werten, die separiert werden durch <Listenzeichen> und den String <supplement> zu dieser Liste (wenn sie noch nicht enthalten sind), aus. Wenn <supplement> die <separator> enthält, können mit diesen Listenzeichen die Einträge in einzelne Strings unterteilt werden und die Prozedur wird für jeden Teilstring angewendet.
Eine typische Verwendung ist der Eintrag zu einer Pfadvariablen, die in der Registry definiert ist.
Supp behält den ursprünglichen Stringtyp (REG_EXPAND_SZ bzw. REG_SZ) bei.
Beispiel:
Der allgemeine Systempfad wird festgelegt durch den Eintrag der Variable Path im Registrierschlüssel
KEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment
Wenn dieser Schlüssel mit OpenKey geöffnet ist, kann mit der Anweisung
supp "Path" ; "C:\utils;%JAVABIN%"
der Pfad ergänzt werden, um die Einträge "C:\utils" sowie "%JAVABIN%".
(Weil der Registry-Eintrag für den Systempfad den Datentyp REG_EXPAND_SZ hat, expandiert Windows %JAVABIN% automatisch zum entsprechenden Verzeichnisnamen, falls %JAVABIN% ebenfalls als Variable definiert ist).

Unter Win2k ist das Phänomen zu beobachten, dass sich der path-Eintrag nur per Skript auslesen (und dann patchen) lässt, wenn vor dem Lesen ein Wert gesetzt wird.

Der alten Wert von Path wird aus der Umgebungsvariable auslesen, wieder in die Registry zurückgeschrieben und dann ist es möglich mit der Registry-Variablen zu arbeiten.

```
[Actions]
DefVar $Path$
set $Path$ = EnvVar ("Path")
Registry_PathPatch

[Registry_PathPatch]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\control\Session Manager\Environment]
set "Path"="$Path$"
supp "Path"; "c:\orawin\bin"
```



Achtung

Nach dem Patchen des Registry-Path enthält die Umgebungsvariable Path den veränderten Wert erst nach einem Reboot oder nach einem Aufruf von [UpdateEnvironment](#) siehe: [\[UpdateEnvironment\]](#)

- **GetMultiSZFromFile** <varname> <Dateiname>
Liest eine Datei zeilenweise in einen Multistring ein und weist diesen <varname> zu.
- **SaveValueToFile** <varname> <filename>
Exportiert die genannten Werte (String oder MultiSZ) in die Datei <filename>
- **DeleteVar** <Varname>
Löscht den Eintrag mit Bezeichnung <Varname> aus dem geöffneten Schlüssel.
- **DeleteKey** <Registrierschlüssel>
Löscht den Registry-Key rekursiv samt aller Unterschlüssel und den enthaltenen Registry-Variablen und -Werten. Zur Syntax, in der der Registrierschlüssel angegeben wird, vgl. OpenKey.

Beispiel:

```
[Registry_KeyLoeschen]
deletekey [HKCU\Environment\subkey1]
```

- **ReconstructFrom** <Dateiname>
(abgekündigt)
- **FlushKey**
Sorgt dafür, dass die Einträge des Schlüssels nicht mehr nur im Speicher gehalten, sondern auf die Platte gespeichert werden (geschieht automatisch beim Schließen eines Keys, insbesondere beim Verlassen einer Registry-Sektion).

10.11.4 Registry-Sektionen, die alle NTUser.dat patchen

Wird eine Registry-Sektion mit dem Parameter `/AllNTUserdats` aufgerufen, so werden ihre Anweisungen für alle auf dem NT-System angelegten User ausgeführt.

Dazu werden zunächst die Dateien NTUser.dat für alle auf dem System eingerichteten User-Accounts durchgegangen (in denen die Registry-Einstellungen aus `HKEY_Users` abgelegt sind). Sie werden temporär in einen Hilfszweig der Registry geladen und dort entsprechenden der Anweisungen der Sektion bearbeitet. Weil dies für den zum Zeitpunkt der Programmausführung angemeldeten User nicht funktioniert, werden die Anweisungen der Sektion zusätzlich für `HKEY_Current_User` ausgeführt. Als Ergebnis verändert sich die gespeicherte NTUser.dat.

Dieser Mechanismus funktioniert nicht für einen angemeldeten User, da seine NTUser.dat in Benutzung ist und der Versuch die Datei zu laden einen Fehler produziert. Damit aber auch für den angemeldeten User Änderungen durchgeführt werden, werden die Registry Kommandos ebenfalls auf den Bereich *HKEY_Current_User* angewendet (*HKEY_Users* ist der Zweig für den angemeldeten Benutzer).

Auch künftig erst angelegte Accounts werden mit erfasst, da auch die NTUser.dat aus dem Profilverzeichnis des *Default Users* bearbeitet wird.

Die Syntax der Sektion ist die einer Standard-Registry-Sektion. Allerdings werden bis vor Version 4.11.2.1 alle Schlüsselnamen relativ interpretiert. D.h. **der Hauptkey ist wegzulassen**: Im folgenden Beispiel werden faktisch die Registry-Einträge für die Variable *FileTransferEnabled* unter *HKEY_Users\XX\Software...* neu hergestellt, sukzessive für alle User auf der Maschine:

```
[Registry_AllUsers]
openkey [Software\ORL\WinVNC3]
set "FileTransferEnabled"=reg_dword:0x00000000
```

Seit *opsi-winst/opsi-script* version 4.11.2 darf man den root key *HKEY_CURRENT_USER* beim *openkey* Kommando mitgeben.

Beispiel:

```
[Registry_AllUsers]
openkey [HKEY_CURRENT_USER\Software\ORL\WinVNC3]
set "FileTransferEnabled"=reg_dword:0x00000000
```

Das hat folgende Vorteile:

- Der Syntax ist leichter verständlich
- Die selbe Registry Sektion kann mit */AllNtuserdats* und in einem *userLoginScript* verwendet werden.

10.11.5 Registry-Sektionen im Regedit-Format

Bei Aufruf von Registry mit dem Parameter */regedit* wird der Inhalt der Registry-Sektion in dem Exportformat erwartet, dass das Standard-Windows-Programm *regedit* erzeugt.

Die von *regedit* generierten Exportdateien haben – von der Kopfzeile abgesehen - den Aufbau von Ini-Dateien haben. Beispiel:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org]

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general]
"bootmode"="BKSTD"
"windomain"=""
"opsiconf"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\shareinfo]
"user"="pcpatch"
"pcpatchpass"=""
"depoturl"="\\\\bonifax\opt_pcb\install"
"configurl"="\\\\bonifax\opt_pcb\pcpatch"
"utilsurl"="\\\\bonifax\opt_pcb\utils"
"utilsdrive"="p:"
"configdrive"="p:"
"depotdrive"="p:"
```

Die Sektionen bezeichnen hier Registry-Schlüssel, die geöffnet werden sollen. Die einzelnen Zeilen stehen für die gewünschten Setzungen von Variablen (entsprechend dem Set-Befehl in *opsi-winst*-Registry-Sektionen).

Diese Anweisungen können aber nun nicht als Sektion innerhalb eine *opsi-winst/opsi-script* Skripts untergebracht werden. Daher kann die Registry Sektion mit dem Parameter */regedit* nur als ausgelagerte Sektion oder über die Funktion *loadTextFile* geladen werden:

```
registry "%scriptpath%/opsiorgkey.reg" /regedit
```

Zu beachten ist noch, dass regedit ab Windows XP nicht mehr das Regedit4-Format produziert, sondern ein Format, das durch die erste Zeile

```
"Windows Registry Editor Version 5.00"
```

gekennzeichnet ist.

Windows sieht hier zusätzliche Wertetypen vor. Gravierender ist, dass die Exportdatei ursprünglich in Unicode erzeugt wird. Um sie mit den 8 Bit-Mitteln der Standardumgebung des *opsi-winst/opsi-script* zu verarbeiten, muss der Zeichensatz konvertiert werden. Die Konvertierung kann z.B. mit einem geeigneten Editor durchgeführt werden. Eine andere Möglichkeit besteht darin, die Konvertierung on the fly vom *opsi-winst/opsi-script* durchführen zu lassen. Dazu lässt sich die String-Listenfunktion `loadUnicodeTextFile` verwenden. Wenn z.B. `printerconnections.reg` ein Unicode-Export ist, wäre regedit in folgender Form aufzurufen:

```
registry loadUnicodeTextFile("%scriptpath%/opsiorgkey.reg") /regedit
```

Auch eine Registry-Patch im regedit-Format kann „für alle NT-User“ ausgeführt werden, sinngemäß in der gleichen Weise wie oben für das gewöhnliche `winst-Registry-Patch-Format` beschrieben. D.h. der Root-Schlüssel `HKCU` muss aus den Angaben entfernt werden und dann wird aus `+ [HKEY_CURRENT_USER\Software\ORL] → [Software\ORL]`.

10.11.6 Registry-Sektionen im AddReg-Format

Die Syntax einer Registry-Sektion, die mit dem Parameter `/addReg` aufgerufen wird, folgt der Syntax von `[AddReg]`-Sektionen in `inf`-Dateien, wie sie z.B. von Treiberinstallationen verwendet wird.

Beispiel:

```
[Registry_ForAcroread]
HKCR, ".fdf", "", 0, "AcroExch.FDFDoc"
HKCR, ".pdf", "", 0, "AcroExch.Document" HKCR, "PDF.PdfCtrl.1", "", 0, "Acr"
```

10.12 OpsiServiceCall Sektion [W/L]

Mit dieser Sektion ist es möglich Informationen abzufragen – oder Daten zu bestimmen – mit Hilfe des opsi Service. Es gibt drei Optionen, mit denen man die Verbindung zum opsi Service definieren kann:

- Per Voreinstellung wird vorausgesetzt, dass das Skript in der Standard opsi Installationsumgebung ausgeführt werden kann. D.h. es besteht eine Verbindung zum opsi Service, die genutzt wird.
- Es wird eine URL für den gewünschten Service und ebenso der benötigte Benutzername und das Passwort als Sektionsparameter gesetzt.
- Es kann ein interaktives Login für den Service gesetzt werden – mit einer voreingestellten Service URL und dem Benutzernamen, wenn das gewünscht wird.

Die abgerufenen Daten können als String-Liste zurückgegeben und dann für die Verwendung in Skripten benutzt werden.

10.12.1 Aufrufparameter



Achtung

Es gibt eine Standard Webserviceverbindung. Diese wird beim Start des opsi-script über den opsi-client-agent auf die bestehende Verbindung zum opsi-server gesetzt.

Werden keine Aufrufparameter angegeben, so wird diese Standardverbindung verwendet. Existiert diese nicht so schlägt der Aufruf fehl.

Es gibt eine Reihe Aufrufparameter, welche eine neue Verbindung aufbauen. **Diese neue Verbindung wird dabei die Standardverbindung. D.h. nachfolgende Aufrufe ohne Aufrufparameter verwenden diese Verbindung solange bis diese wieder explizit verändert wird, oder das Produktskript abgearbeitet ist.**

Ein neues Produkt fängt wieder mit der ursprünglichen Webservice Verbindung an.

Aufrufparameter welche die Standardverbindung verändern:

- /interactive
- /serviceurl /username /password
- /opsiclientd

Wiederherstellung der ursprünglichen Verbindung:

Über den Aufruf einer opsiServiceCall Sektion mit dem Aufrufparameter /preloginservice wird die Standardverbindung wieder auf den ursprünglichen Wert zurückgesetzt. Alternativ kann auch der Sektionsfreie Aufruf:

```
opsiServiceCall /preloginservice
```

verwendet werden.

Die Aufrufparameter:

Es gibt Optionen, mit denen man die Verbindung zu einem opsi Service angeben kann und Einstellungen, die für die Verbindung benötigt werden.

Verbindungsparameter können mit Hilfe von

- /serviceurl <url to the opsi web service>
- /username <web service user name>
- /password <web service user password>

gesetzt werden. Wenn diese Parameter definiert sind (oder zumindest einer der Parametern), wird versucht eine Verbindung zu der genannten Service URL herzustellen und bei Erfolg diese zur Standardverbindung zu machen.

Die Option

- /interactive
bedeutet, dass der Benutzer die Verbindungsdaten bestätigen muss und das Passwort eingibt. Diese Option kann damit nicht in Skripten verwendet werden, die voll automatisch ausgeführt werden sollen.
- /preloginservice
setzt die Standardverbindung wieder auf den beim Start gesetzten Wert zurück.
- /opsiclientd //since 4.11.2.1
ruft den Webservice des lokalen opsiclientd auf und verändert die Standardverbindung.
- /opsiclientd-once //since 4.11.6.11
ruft den Webservice des lokalen opsiclientd auf und setzt nach dem Aufruf die Standardverbindung wieder auf den ursprünglichen Wert zurück.

10.12.2 Sektionsformat

Ein `opsiServiceCall`, welcher eine existierende Verbindung zu einem opsi Service benutzt, wird bestimmt durch den Methodennamen und eine Parameterliste.

Beide werden in dem Sektionsabschnitt definiert und haben folgendes Format:

```
"method":<method name>
"params": [
  <params>
]
```

Dabei sind `<params>` kein, ein oder auch mehrere durch Komma getrennte Strings. Welche Parameter benötigt werden, hängt von der aufgerufenen Methode ab.

Beispiel:

```
[opsiservicecall_clientIdsList]
"method": "getClientIds_list"
"params": []
```

Die Sektion erstellt eine Liste der PC-Namen (IDs) von allen lokalen opsi Benutzern. Wenn es für andere Zwecke als Test und Dokumentation genutzt werden soll, kann die Sektion als ein Teil eines String-Listen Ausdrucks (vgl. das folgende Beispiel) verwendet werden.

```
DefStringList $result$
Set $result$=getReturnListFromSection("opsiservicecall_clientIdsList")
```

Die Verwendung von `getReturnListFromSection` ist dokumentiert in dem Kapitel zur String-Listenverarbeitung dieses Handbuchs (siehe [Kapitel "String-Listen-Erzeugung mit Hilfe von Sektionsaufrufen"](#)).

Ein Hash, der eine Namensliste mit Wertepaaren enthält, wird durch den folgenden opsi Service aufgerufen (beinhaltet keine leere Parameterliste):

```
[opsiservicecall_hostHash]
"method": "getHost_hash"
"params": [
  "pcbon8.uib.local"
]
```

Objekt orientierte Methoden Der Umgang mit JSON Objekten aus dem Webservice erfordert ein Grundverständnis von JSON, den opsi Objekten und den JSON bezogenen Methoden in opsi-script. Siehe dazu auch : opsi-manual: Kapitel: "Web service / API Methoden seit opsi 4.0" In diesem Manual: ####

Gemäß folgendem Code Beispiel können Sie Objekte vom Service holen. In diesem Beispiel werden Alle productOnClient Objekte geholt, welche zum aktuellen Rechner gehören (`%opsiserviceUser%` ist im Service Kontext der FGDN des Clients), Localboot Produkte sind und bei denen der Actionrequest auf `setup` steht.

```
DefStringlist $resultlist$
set $resultlist$ = getReturnListFromSection("opsiServiceCall_get_productOnClient_setup_objects")
[opsiServiceCall_get_productOnClient_setup_objects]
"method": "productOnClient_getObjects"
"params": [
  "[]",
  '{"clientId": "%opsiserviceUser%", "productType": "LocalbootProduct", "actionRequest": "setup"}',
]
```

Das Ergebnis ist ein JSON Array String welcher in der ersten Zeile von `$resultlist$` steht.

Sie können auch (veränderte) Objekte wieder zurückschreiben. Folgendes Beispiel zeigt das Prinzip: Die String Variable `$ArrayStr$` muss ein gültiges JSON Array enthalten.

```

DefVar $ArrayStr$
(...)
[opsiServiceCall_updatePOC]
"method": "productOnClient_updateObjects"
"params": [
    '$ArrayStr$'
]

```

10.12.3 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-script-test* und dort den Bereich `$Flag_winst_opsiServiceCall$ = "on"`

10.13 ExecPython Sektionen [W/L]

Die ExecPython Sektionen basieren auf Shell-Sektionen (ähnlich wie `DosInAnIcon`). Während diese den Inhalt der Sektion dem Interpreter `cmd.exe` übergeben, wird der Inhalt einer ExecPython Sektion dem Python Interpreter übergeben (welcher auf dem System installiert sein muss).

Beispiel

Das folgende Beispiel demonstriert einen `execPython` Aufruf mit einer Parameterliste zu dem `print` Python-Kommando.

Der Aufruf könnte wie folgt aussehen

```

execpython_hello -a "option a" -b "option b" "there we are"

[execpython_hello]
import sys
print "we are working in path: ", a
if len(sys.argv) > 1 :
    for arg in sys.argv[1:] :
        print arg
else:
    print "no arguments"

print "hello"

```

Die Ausgabe des Druck-(print) Kommandos wird gesammelt und in einer Logdatei geschrieben. So kann man die folgende Logdatei bekommen

```

output:
-----
-a
option a
-b
option b
there we are
    hello

```

Anzumerken ist hierbei, dass der `loglevel` auf `1` gesetzt werden muss, damit die Ausgabe wirklich den Weg in die Logdatei findet.

10.13.1 Verflechten eines Python Skripts mit einem *opsi-winst/opsi-script* Skript

Aktuell ist die `execPython` Sektion dem *opsi-winst/opsi-script* Skript über vier Kategorien von gemeinsam genutzten Daten integriert:

- Eine Parameterliste geht zum Python Skript über.
- Alles was vom Python Skript gedruckt wird, wird in die *opsi-winst/opsi-script* log-Datei geschrieben.
- Der *opsi-winst/opsi-script* Skript Mechanismus für die Einführung von Konstanten und Variablen in Sektionen arbeitet erwartungsgemäß für die `execPython` Sektion.
- Die Ausgabe einer `execPython` Sektion kann umgewandelt werden in eine String-Liste und dann vom laufenden *opsi-winst/opsi-script* Skript weiter verwendet werden.

Ein Beispiel für die ersten beiden Wege der Verflechtung des Python Skripts mit dem *opsi-winst/opsi-script* Skript werden im Anschluss beschrieben. Es wurde erweitert, damit einige der Werte von *opsi-winst/opsi-script* Konstanten oder Variablen aufgerufen werden können.

```
[execpython_hello]
import sys
a = "%scriptpath%"
print "we are working in path: ", a
print "my host ID is ", "%hostID%"
if len(sys.argv) > 1 :
    for arg in sys.argv[1:] :
        print arg
else:
    print "no arguments"

print "the current loglevel is ", "$loglevel$"
print "hello"
```

Allerdings muss die `$loglevel$` Variable vor dem Aufruf der `ExecPython` Sektion gesetzt werden:

```
DefVar $LogLevel$
set $loglevel$ = getLogLevel
```

Damit wir am Ende in der Lage sind, die Ergebnisse der Ausgabe weiter zu verarbeiten, wird eine String-List Variable erstellt, die über die `execPython` Sektion folgendermaßen aufgerufen werden kann:

```
DefStringList pythonresult
Set pythonResult = GetOutputStreamFromSection('execpython_hello -a "opt a"')
```

10.13.2 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich `$Flag_compare_to_python$ = "on"`

10.14 ExecWith Sektionen [W/L]

ExecWith Sektionen sind verallgemeinerte *DosBatch* bzw. *ExecPython* Sektionen: Welches Programm den Inhalt der Sektionen ausführt, wird durch einen Parameter beim Sektionsaufruf bestimmt.

Wenn der Aufruf so lautet:

```
'execPython_hello -a "hello" -b "world"'
```

so sind

```
-a "hello" -b "world"
```

Parameter, die vom Pythonskript akzeptiert werden. Mit dem ExecWith Aufruf sieht der gleiche Ausdruck wie folgt aus:

```
'execWith_hello "python" PASS -a "hello" -b "world" WINST /EscapeStrings'
```

Die Option `/EscapeStrings` wird in der ExecPython-Sektion automatisch angewendet und bedeutet, dass Backslashes und Konstanten in String-Variablen dupliziert werden, bevor sie das aufgerufene Programm interpretiert.

10.14.1 Aufrufparameter (Modifier)

Generell haben wir die Aufrufsyntax:

```
ExecWith_SECTION PROGRAM PROGRAMPARAS pass PASSPARAS winst WINSTOPTS
```

Jeder der Ausdrücke *PROGRAM*, *PROGRAMPARAS*, *PASSPARAS*, *WINSTOPTS* können beliebige String-Ausdrücke oder auch einfache String-Konstanten (ohne Anführungszeichen) sein.

Die Schlüsselwörter `PASS` und `WINST` dürfen fehlen, wenn der entsprechende Part nicht existiert.

Es sind folgende *opsi-winst/opsi-script*-Optionen verfügbar:

- `/32Bit`
Das ist der Default. Der angegebene Interpreterpfad wird als 32 Bit Pfad interpretiert.
Beispiel: `c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe` ruft (auch auf einem 64bit System) die 32 Bit `powershell.exe` auf.
- `/64Bit`
Der angegebene Interpreterpfad wird als 64 Bit Pfad interpretiert.
Beispiel: `c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe` ruft (auf einem 64bit System) die 64 Bit `powershell.exe` auf.
- `/SysNative`
Der angegebene Interpreterpfad wird gemäß der OS Architektur interpretiert.
Beispiel: `c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe` ruft auf einem 64bit System die 64 Bit `powershell.exe` und auf einem 32bit System die 32 Bit `'powershell.exe'` auf.
- `/EscapeStrings`
Diese Option legt fest, dass die Backslashes in *opsi-winst/opsi-script*-Variablen und Konstanten dupliziert werden, so dass sie das ausführende Programm in der üblichen Form von Strings in *C*-Syntax vorfindet.
- `/LetThemGo`
Diese Option hat den Effekt (wie bei `winBatch` Aufrufen), dass das aufgerufene Programm in einem neuen Thread startet, während der *opsi-winst/opsi-script* mit dem Auslesen des Skripts fortfährt.

Wie bei ExecPython Sektionen wird die Ausgabe einer ExecWith-Sektion in einer String-Liste über die Funktion `getOutputStreamFromSection` erfasst.

Der Inhalt der Sektion wird in eine temporäre Datei (*.bat) gespeichert. Seit Version 4.11.3.5 wird, wenn im angegebene Interpreterpfad `powershell.exe` vorkommt, die temporäre Datei als `.ps1` gespeichert.

Note

Die Ausführung von Skripten ist in der Powershell per default ausgeschaltet. Um eine Execwith Sektion mit powershell nutzen zu können, muß die Skriptausführung erst erlaubt werden. Dazu führen Sie `powershell.exe set-executionpolicy RemoteSigned` in einer `DosInAnIcon` Sektion aus.

Beispiel

```

DosInAnIcon_setpolicy
ExecWith_powershell powershell.exe
set $exitcode$ = getLastExitcode
if not ($exitcode$ = "0")
    comment "powershell script failed"
endif

[DosInAnIcon_setpolicy]
echo "powershell set-executionpolicy RemoteSigned ..."
powershell.exe set-executionpolicy RemoteSigned
exit %ERRORLEVEL%

[ExecWith_powershell]
echo "powershell opsi-script-test"
if ($?) {Exit(0)}
else {Exit(1)}

```

10.14.2 Weitere Beispiele

Der folgende Aufruf verweist auf eine Sektion, die ein *autoit3*-Skript ist, das auf zu öffnende Fenster wartet (dafür ist die Option `/letThemGo` zu benutzen), um sie dann in der aufgerufenen Reihenfolge zu schließen:

```
ExecWith_close "%SCRIPTPATH%\autoit3.exe" WINST /letThemGo
```

Ein einfacher Aufruf

```
ExecWith_edit_me "notepad.exe" WINST /letThemGo
```

ruft Notepad auf und öffnet die Sektion als Datei (allerdings ohne die Zeilen die mit einem Semikolon beginnen, da der *opsi-winst/opsi-script* solche Zeilen als Kommentarzeilen interpretiert und vor der weiteren Behandlung der Sektion entfernt).

Für zusätzliche Beispiele beachten Sie das Produkt *opsi-script-test* und dort den Bereich `$Flag_autoit3_test$ = "on"`.

10.15 LDAPsearch Sektion [W]

Eine LDAPsearch Sektion beschreibt eine Suchanfrage an ein LDAP Verzeichnis, die ausgeführt wird und auch die Antwort empfängt (und wenn möglich im Cache speichert).

Bevor wir zu den *opsi-winst/opsi-script* Kommandos übergehen, gibt es erst noch einige Erklärungen zum Syntax von LDAP selbst

10.15.1 LDAP – Protokoll, Service, Verzeichnis

LDAP bedeutet "Lightweight Directory Access Protocol" und ist, wie der Name besagt, ein festgelegter Weg der Kommunikation mit einem Datenverzeichnis.

Dieses Verzeichnis ist für gewöhnlich hierarchisch organisiert. Es ist eine hierarchische Datenbank oder ein Datenbaum.

Ein **LDAP Service** implementiert das Protokoll zum Lesen und Schreiben auf diesem Verzeichnis. Ein Verzeichnis, das über einen LDAP Service angesteuert werden kann, nennt sich **LDAP directory**.

Für ein Beispiel werfen einen Blick auf einen Bereich eines LDAP Verzeichnisbaums mit Daten aus dem opsi LDAP-Backend (angezeigt im Open Source LDAP-Browser JXPlorer).

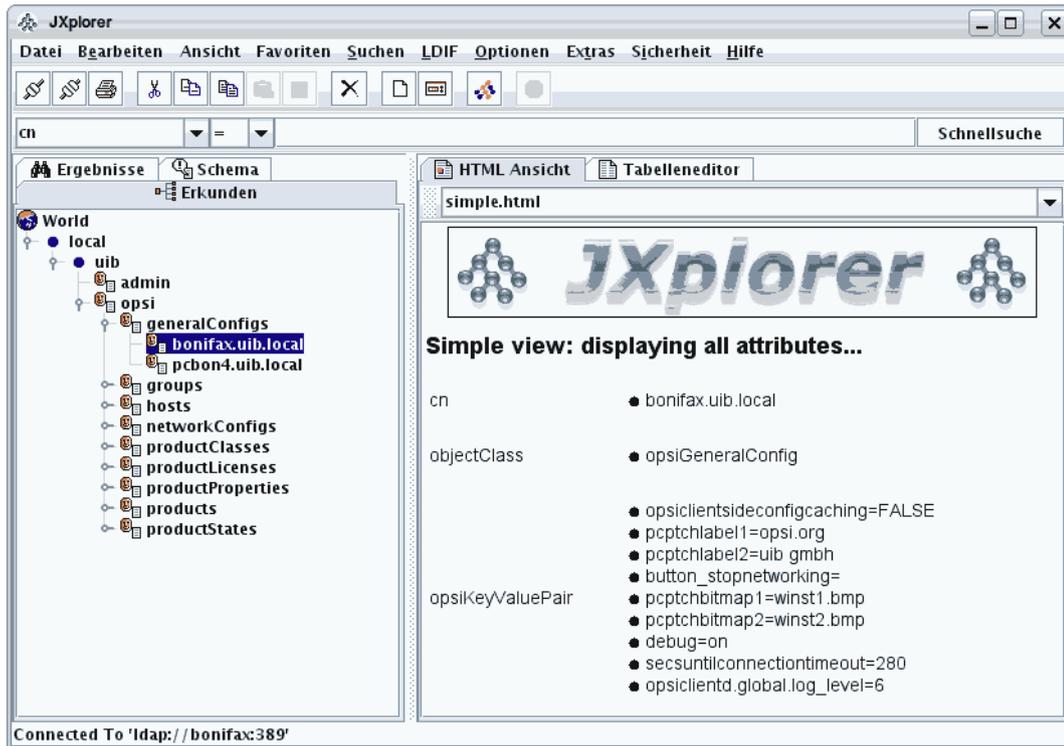


Abbildung 10.4: Ansicht von verschiedenen Bereichen des opsi LDAP Baums

Ein **LDAP search request** ist ein Suchabfrage an das LDAP Verzeichnis über einen LDAP Service. Als Antwort werden verschiedene Inhalte des Datenverzeichnisses zurückgegeben.

Grundsätzlich beschreiben Suchabfragen den Pfad im Verzeichnisbaum, der zu der gewünschten Information führt. Der Pfad ist der **distinguished name** (dn) zusammengesetzt aus den Namen der Knoten ("relative distinguished names") welche den Pfad bilden. Zum Beispiel:

local/uib/opsi/generalConfigs/bonifax.uib.local

Da jeder Knoten als eine Instanz einer strukturellen Objektklasse konzipiert ist, wird die Pfadbeschreibung in folgender Form ausgegeben: mit Klassentyp (und beginnend mit dem letzten Pfadelement):

cn=bonifax.uib.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local

Der Pfad in einer Abfrage muss nicht notwendigerweise „komplett“ sein und auch nicht zu einem einzelnen Blatt (Teil) des Baumes führen. Im Gegenteil, unvollständige Pfade sind üblich.

Aber auch wenn der Pfad zu einem einzelnen Blatt führt, kann dieses wiederum mehrere Werte enthalten. Jeder Knoten des Baumes hat eine oder mehrere Klassen als Attributtypen. Zu jeder Klasse können ein oder mehrere Werte zugehörig sein.

Bei einem gegebenen Abfragepfad könnten wir uns interessieren für

1. für die Knoten – auch LDAP Objekte genannt – zu welchen der Pfad führt,
2. für die Attribute, die zu den Knoten gehören,
3. und die Werte, die sowohl zu den Objekten wie zu den Attributen gehören.

Offensichtlich ist der Umgang mit der Fülle der Informationen möglicher Antworten die vorrangige Herausforderung bei der Abwicklung von LDAP Abfragen.

Der folgende Abschnitt zeigt eine LDAP Abfrage über den Bereich des LDAP Baums, welcher in der obenstehenden Grafik abgebildet ist.

Beispiel einer LDAP Antwort

Eine *opsi-winst/opsi-script* Sektion `ldapsearch_generalConfigs` ist wie folgt definiert:

```
[ldapsearch_generalConfigs]
targethost: bonifax
dn: cn=generalConfigs,cn=opsi,dc=uib,dc=local
```

Der Sektionsaufruf gibt eine LDAP Antwort zurück, die folgendermaßen aussieht:

```
Result: 0
  Object: cn=generalConfigs,cn=opsi,dc=uib,dc=local
  Attribute: cn
    generalConfigs
  Attribute: objectClass
    organizationalRole
Result: 1
  Object: cn=pcbon4.uib.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
  Attribute: cn
    pcbon4.uib.local
  Attribute: objectClass
    opsiGeneralConfig
  Attribute: opsiKeyValuePair
    test2=test
    test=a b c d
Result: 2
  Object: cn=bonifax.uib.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
  Attribute: objectClass
    opsiGeneralConfig
  Attribute: cn
    bonifax.uib.local
  Attribute: opsiKeyValuePair
    opsiclientsideconfigcaching=FALSE
    pcptchlabel1=opsi.org
    pcptchlabel2=uib gmbh
    button_stopnetworking=
    pcptchbitmap1=winst1.bmp
    pcptchbitmap2=winst2.bmp
    debug=on
    secsuntilconnectiontimeout=280
    opsiclientd.global.log_level=
```

Es gibt nun verschiedene *opsi-winst/opsi-script* Optionen, um die Komplexität der Auswertung der Ergebnisse solcher Anfragen zu reduzieren und zu handhaben.

10.15.2 LDAPsearch Aufrufparameter

Für den Aufruf von LDAPSearch Sektionen sind zwei Typen von Optionen definiert.

- cache options
- output options

Die *cache options* sind:

- /cache
- /cached
- /free
- (no cache option)

Wenn keine cache Option spezifiziert wurde, wird die Antwort der LDAP Suche nicht für zukünftige Anwendung gespeichert.

Bei der `/cache` Option wird die Antwort für zukünftige Auswertungen gespeichert, die `/cached` Option verweist auf die letzte gespeicherte Antwort, welche wiederverwendet wird, statt eine neue Suche zu starten, die `/free` Option löscht die gecachten Antworten (dies ist vor allem bei Suchanfragen mit sehr langen Antworten sinnvoll).

Die output options sind:

- `/objects`
- `/attributes`
- `/values`
- (no output option)

Die Ausgabeoptionen bestimmen die String-Listen, die produziert werden, wenn eine LDAPsearch Sektion über `getReturnlistFromSection` aufgerufen wird:

- Wenn die Ausgabeoptionen nicht näher spezifiziert werden, wird die komplette LDAP Antwort aufgelistet.
- Die Optionen `objects`, `attributes` und `values` beschränken die Ausgabe entsprechend auf Zeilen zu Objekten, Attributen bzw. Werten in der LDAP Antwort.

Zu beachten ist, dass die ausgegebenen Listen von Attributen nur dann dem richtigen Objekt zu geordnet werden können, wenn die gesamte Ausgabe nur noch ein Objekt enthält. Ebenso sind Werte nur dann dem korrekten Attribut zuordenbar, wenn nur noch ein Attribut in der Ausgabeliste vorkommt.

Daher wird so vorgegangen, dass eine ursprüngliche Suche immer weiter eingeeengt wird bis nur noch ein Objekt bzw. Attribut zurückgegeben wird. Dies kann über entsprechende Count Aufrufe überprüft werden.

Die Einengung der ursprünglichen Suche geht sehr schnell, wenn diese auf der gecachten Antwort durchgeführt wird.

10.15.3 Einengung der Suche

Ein Beispiel soll zeigen, wie die Suche soweit eingeschränkt werden kann, damit ein bestimmtes Ergebnis bei einer Suche im LDAP Verzeichnis erreicht werden kann.

Wir starten mit dem Aufruf von `ldapsearch_generalConfigs` (wie oben beschrieben), fügen den cache Parameter hinzu, `ldapsearch_generalconfigs /cache`

die Abfrage wird ausgeführt und die Antwort für zukünftige Nutzung gespeichert.

Dann gibt der Aufruf

```
getReturnlistFromSection("ldapsearch_generalconfigs /cached /objects")
```

folgende Liste aus

```
cn=generalconfigs,cn=opsi,dc=uib,dc=local
cn=pcbon4.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
cn=bonifax.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
```

Wenn wir die Auswahl im Baumverzeichnis mit

```
[ldapsearch_generalConfigs]
targethost: bonifax
dn: cn=bonifax.ubi.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
```

einschränken und nochmal starten, enthält die Objektliste nur noch folgende Einträge

```
cn=bonifax.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
```

Die dazugehörige Attributliste enthält drei Elemente:

```
objectclass
cn
opsikeyvaluepair
```

Um die zugehörigen Werte zu einem einzelnen Attribut zu bekommen, muss die Abfrage noch erweitert werden:

```
[ldapsearch_generalConfigs]
targethost: bonifax
dn: cn=bonifax.uib.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
attribute: opsiKeyValuePair
```

Das Ergebnis ist eine Attributliste, die nur ein Element enthält. Die Liste mit den zugehörigen Werten sieht wie folgt aus

```
opsiclientsideconfigcaching=false
pcptchlabel1=opsi.org
pcptchlabel2=uib gmbh
button_stopnetworking=
pcptchbitmap1=winst1.bmp
pcptchbitmap2=winst2.bmp
debug=on
secsuntilconnectiontimeout=280
opsiclientd.global.log_level=6
```

Es gibt keine LDAP Mittel um diese Ergebnis noch weiter einzugrenzen!

(Aber die *opsi-winst/opsi-script* Funktion `getValue (key, list)` (vgl. [Kapitel "\(Wieder-\) Gewinnen von Einzelstrings aus String-Listen"](#)) hilft in diesem Fall: z.B. `getValue ("secsuntilconnectiontimeout", list)` würde die gewünschte Zahl ausgeben).

Mit der Funktion `count (list)` kann überprüft werden, ob die Eingrenzung der Suchabfrage erfolgreich war. In den meisten Fällen ist gewünscht, dass das Ergebnis "1" ist.

10.15.4 LDAPsearch Sektion Syntax

Eine LDAPsearch Sektion beinhaltet die Spezifikationen:

- **targethost:**
Der Server, der das LDAP Verzeichnis verwaltet/gespeichert wird (service), muss benannt werden.
- **targetport:**
Wenn der Port eines LDAP Service nicht der voreingestellte Port von 389, muss er an dieser Stelle angegeben werden. Wenn die Spezifizierung nicht erfolgt, wird der Default-Port verwendet.
- **user:**
Zu verwendender user Name. Seit 4.11.3.5
- **password:**
Zu verwendendes user Passwort. Seit 4.11.3.5
- **dn:**
Hier kann der charakteristische Name (distinguished name), der „Suchpfad“, für die Suchanfrage gegeben werden.
- **typesonly:**
Per Voreinstellung ist der Wert "false", was bedeutet das auch die Werte ermittelt werden.

- **filter:**
Der Filter für eine LDAP Suche hat eine spezielle LDAP Syntax, die nicht vom *opsi-winst/opsi-script* überprüft wird. Voreingestellt ist "(objectclass=*)".
- **attributes:**
Durch Kommas werden die Attributnamen in einer Liste getrennt. Die Default-Einstellung ist eine Liste, in der alle Attribute aufgeführt werden.

10.15.5 Beispiele

Ein kurzes und sehr realistisches Beispiel soll am Ende dieses Abschnittes aufgeführt werden:

\$founditems\$ ist eine StringList Variable und *\$opsiClient\$* ist eine String-Variable. Der Aufruf von *getReturnlistFromSection* liefert die Ergebnisse. Das nachfolgende Codefragment gibt das eindeutige Ergebnis für *\$opsiDescription\$* zurück, wenn dieses existiert. Es vermeldet einen Fehler, wenn die Suche ein unerwartetes Ergebnis zurück gibt:

```
set $opsiClient$ = "test.uib.local"
set $founditems$ = getReturnlistFromSection("ldapsearch_hosts /values")

DefVar $opsiDescription$
set $opsiDescription$ = ""
if count(founditems) = "1"
    set $opsiDescription$ = takeString(0, founditems)
else
    if count(founditems) = "0"
        comment "No result found"
    else
        logError "No unique result for Ldapsearch for client " + $opsiClient$
    endif
endif

[ldapsearch_hosts]
targethost: opsiserver
targetport:
dn: cn=$opsiClient$,cn=hosts,cn=opsi,dc=uib,dc=local
typesOnly: false
filter: (objectclass=*)
attributes: opsiDescription
```

Beispiel mit user / password

```
comment ""
comment "-----"
comment "Testing: "
comment "user / password"
Set $LdapHost$ = "vmix7.uib.local"
Set $LdapPort$ = "389"
Set $LdapUser$ = "cn=Administrator,cn=Users,dc=uib,dc=local"
Set $LdapPassword$ = "Linux123"
Set $LdapResultType$ = "objects"
Set $LdapSearchDn$ = "cn=Users,dc=uib,dc=local"
Set $LdapSearchAttributes$ = "name,objectClass"
Set $LdapFilter$ = "(&(objectclass=*))"

markErrorNumber
set $list1$ = getReturnListFromSection("ldapsearch_users /" + $LdapResultType$)
if errorsOccurredSinceMark > 0
```

```
        comment "failed while ldapsearch"
        set $TestResult$ = "not o.k."
else
        comment "passed"
endif

[ldapsearch_users]
targethost: $LdapHost$
targetport: $LdapPort$
user: $LdapUser$
password: $LdapPassword$
dn: $LdapSearchDn$
attributes: $LdapSearchAttributes$
filter: $LdapFilter$
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich `$Flag_winst_ldap_search$ = "on"`.

Kapitel 11

64 Bit-Unterstützung unter Windows [W]

Der *opsi-winst/opsi-script* ist ein 32 Bit-Programm. Damit sich auch 32 Bit-Programme auf 64 Bit-Systemen normal arbeiten können, gibt es für 32 Bit-Programme sowohl in der Registry als auch im Dateisystem Spezialbereiche auf die Zugriffe umgeleitet werden, die sonst in 64 Bit vorbehaltenen Bereichen landen würden.

So wird ein Zugriff auf `c:\windows\system32` umgelenkt auf `c:\windows\syswow64`.

Aber ein Zugriff auf `c:\program files` wird **nicht** umgelenkt auf `c:\program files (x86)`

So wird ein Registry Zugriff auf `[HKLM\software\opsi.org]` umgelenkt auf `[HKLM\software\wow6432node\opsi.org]`.

opsi-winst installiert daher als 32 Bit-Programm Skripte, die unter 32 Bit laufen, auch in 64 Bit-Systemen korrekt.

Für die Installation von 64 Bit-Programmen liefern einige alte Konstanten wie `'%ProgramFilesDir%'` für 64 Bit-Programme die falschen Werte. Daher gibt es ab winst Version 4.10.8 folgende Neuerungen:

In der Regel kann (und sollte) nun explizit angegeben werden, wohin geschrieben und woher gelesen werden soll. Dazu gibt es drei Varianten:

32

Explizit 32 Bit

64

Explizit 64 Bit. Wenn es das nicht gibt, dann architekturenspezifisch.

SysNative

Entsprechend der Architektur auf dem das Skript läuft.

Entsprechend gibt es zusätzlichen Konstanten:

Tabelle 11.1: Konstanten

Konstante	32 Bit	64 Bit
<code>%ProgramFilesDir%</code>	<code>c:\program files</code>	<code>c:\program files (x86)</code>
<code>%ProgramFiles32Dir%</code>	<code>c:\program files</code>	<code>c:\program files (x86)</code>
<code>%ProgramFiles64Dir%</code>	<code>c:\program files</code>	<code>c:\program files</code>
<code>%ProgramFilesSysnativeDir%</code>	<code>c:\program files</code>	<code>c:\program files</code>

`%ProgramFilesDir%`

sollte in Zukunft besser gemieden werden.

%ProgramFiles32Dir%

sollten Sie verwenden, wenn Sie explizit 32 Bit-Software installieren wollen.

%ProgramFiles64Dir%

sollten Sie verwenden, wenn Sie explizit 64 Bit-Software installieren wollen.

%ProgramFilesSysnativeDir%

sollten Sie verwenden, wenn Sie auf den Default der Architektur zugreifen wollen.

Für den Zugriff auf eigentlich 64 Bit-Software vorbehaltene Bereiche kennt der *opsi-winst/opsi-script* folgende zusätzlichen Befehle:

- `GetRegistrystringValue32`
- `GetRegistrystringValue64`
- `GetRegistrystringValueSysNative`
- `FileExists32`
- `FileExists64`
- `FileExistsSysNative`

Bei den folgenden Funktionen kann die Zugriffsart als Parameter mitgegeben werden (der default ist `sysnative`):

- `getRegistryValue`
- `RegKeyExists`
- `RegVarExists`
- `powershellCall`

Registry-Sektionen schreiben in den 32 Bit-Bereich der Registry. Ebenfalls werden in Files-Sektionen Zugriffe auf `c:\windows\system32` umgelenkt.

Für Registry, Files und Winbatch Sektionen gibt es daher nun die Aufrufparameter:

- `/32Bit`
Das ist der Default. Schreibzugriffe werden in die 32 Bit-Registry bzw. das 32 Bit-Systemverzeichnis gelenkt.
- `/64Bit`
Schreibzugriffe werden in die 64 Bit-Registry bzw. das 64 Bit-Systemverzeichnis gelenkt. Gibt es diesen nicht, so wird der architektur-spezifische Zweig verwendet.
- `/SysNative`
Schreibzugriffe werden in den architektur-spezifischen Zweig der Registry bzw. des Systemverzeichnisses gelenkt.

Für `DosBatch`, `DosInAnIcon` (`ShellBatch`, `ShellInAnIcon`) sowie `Execwith` gilt das selbe, nur das die Parameter durch das Schlüsselwort `winst` abgegrenzt werden müssen.

Beispiel:

```
DosInAnIcon_do_64bit_stuff winst /64Bit
```

Als weitere Möglichkeit für explizite 64 Bit-Operationen wird bei der Installation des opsi-client-agent die Datei `c:\windows\system32\cmd.exe` nach `c:\windows\cmd64.exe` kopiert. Durch den Aufruf von Skripten mit dieser `cmd64.exe` im Rahmen von `ExecWith` Sektionen können beliebige 64 Bit-Operationen ausgeführt werden.

Beispiele:

File handling:

```

if $INST_SystemType$ = "64 Bit System"
    comment ""
    comment "-----"
    comment "Testing: "
    message "64 Bit redirection"
    Files_copy_test_to_system32
    if FileExists("%System%\dummy.txt")
        comment "passed"
    else
        LogWarning "failed"
        set $TestResult$ = "not o.k."
    endif
    ExecWith_remove_test_from_system32 'cmd.exe' /C
    Files_copy_test_to_system32 /64Bit
    if FileExists64("%System%\dummy.txt")
        comment "passed"
    else
        LogWarning "failed"
        set $TestResult$ = "not o.k."
    endif
    ExecWith_remove_test_from_system32 '%SystemRoot%\cmd64.exe' /C
endif

```

Registry Handling:

```

message "Write to 64 Bit Registry"
if ($INST_SystemType$ = "64 Bit System")
    set $ConstTest$ = ""
    set $regWriteValue$ = "64"
    set $CompValue$ = $regWriteValue$
    Registry_opsi_org_test /64Bit
    ExecWith_opsi_org_test "%systemroot%\cmd64.exe" /c
    set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
bitByWinst")
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        comment "failed"
    endif
    set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
bitByReg")
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        comment "failed"
    endif
    set $regWriteValue$ = "32"
    set $CompValue$ = $regWriteValue$
    Registry_opsi_org_test
    ExecWith_opsi_org_test "cmd.exe" /c
    set $ConstTest$ = GetRegistryStringValue("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
bitByWinst")
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else

```

```

        set $TestResult$ = "not o.k."
        comment "failed"
    endif
    set $ConstTest$ = GetRegistryStringValue("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
bitByReg")
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        comment "failed"
    endif
endif
else
    set $regWriteValue$ = "32"
    set $CompValue$ = $regWriteValue$
    Registry_opsi_org_test /64Bit
    ExecWith_opsi_org_test "cmd.exe" /c
    set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
bitByWinst")
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        comment "failed"
    endif
    set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
bitByReg")
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        comment "failed"
    endif
endif
endif

if ($INST_SystemType$ = "64 Bit System")
    set $regWriteValue$ = "64"
    Registry_hkcu_opsi_org_test /AllNtUserDats /64Bit
    set $regWriteValue$ = "32"
    Registry_hkcu_opsi_org_test /AllNtUserDats
else
    set $regWriteValue$ = "32"
    Registry_hkcu_opsi_org_test /AllNtUserDats
    Registry_hkcu_opsi_org_test /AllNtUserDats /64Bit
endif

```

Kapitel 12

Kochbuch

In diesem Kapitel sind Skript-Beispiele zusammengestellt, wie durch den Einsatz verschiedener *opsi-winst/opsi-script* Funktionen gewisse Aufgaben, die sich in ähnlicher Weise immer wieder stellen, bewältigt werden können.

12.1 Löschen einer Datei in allen Userverzeichnissen

Seit *opsi-winst/opsi-script* Version 4.2 gibt es für diese Aufgabe eine einfache Lösung: Wenn etwa die Datei `alt.txt` aus allen Userverzeichnissen gelöscht werden soll, so kann der folgende Files-Sektions-Aufruf verwendet werden:

```
files_delete_Alt /allNtUserProfiles

[files_delete_Alt]
delete "%UserProfileDir%\alt.txt"
```

Für ältere *opsi-winst/opsi-script* Versionen sei hier noch ein Workaround dokumentiert, der hilfreiche Techniken enthält, die eventuell für andere Zwecke dienen können. Folgende Zutaten werden benötigt:

- Eine `DosInAnIcon`-Sektion, in der ein `dir`-Befehl die Liste aller Verzeichnisnamen produziert.
- Eine Files-Sektion, die das Löschen der Datei `alt.txt` in einem bestimmten Verzeichnis anstößt.
- Eine String-Listen-Verarbeitung, die alles miteinander verknüpft.

Das Ganze kann z.B. so aussehen:

```
[Actions]

; Variable für den Dateinamen:
DefVar $loeschDatei$
set $loeschDatei$ = "alt.txt"

; Variablendeklaration für die String-Listen
DefStringList list0
DefStringList list1

; Einfangen der vom Dos-dir-Befehl produzierten Zeilen
Set list0 = getOutputStreamFromSection ('dosbatch_profiledir')

; Aufruf einer Files-Sektion für jede Zeile
for $x$ in list0 do files_delete_x
```

```
; Und hier die beiden benötigten Spezialsektionen:
[dosbatch_profiledir]
dir "%ProfileDir%" /b

[files_delete_x]
delete "%ProfileDir%\$x$\$LoeschDatei$"
```

12.2 Überprüfen, ob ein spezieller Service läuft

Wenn wir überprüfen wollen, ob ein spezieller Service (beispielsweise der "opsiclientd") läuft und ihn, falls er nicht läuft, starten wollen, müssen wir folgendes Skript verwenden.

Um eine Liste der laufenden Services angezeigt zu bekommen, müssen wir das Kommando

```
net start
```

in einer DosBatch Sektion starten und das Ergebnis in der \$list0\$ erfassen. Wir gleichen die Liste ab und iterieren die Elemente, um zu sehen, ob der spezielle Service beinhaltet ist. Wenn er nicht da ist, wird er gestartet.

```
[Actions]
DefStringList $list0$
DefStringList $list1$
DefStringList $result$
Set $list0$=getOutputStreamFromSection('DosBatch_netcall')
Set $list1$=getSublist(2:-3, $list0$)

DefVar $myservice$
DefVar $compareS$
DefVar $splitS$
DefVar $found$
Set $found$ = "false"
set $myservice$ = "opsiclientd"

comment "====="
comment "search the list"
; for developping loglevel = 7
; setloglevel=7
; in normal use we dont want to log the looping
setloglevel = 5
for %s% in $list1$ do sub_find_myservice
setloglevel=7
comment "====="

if $found$ = "false"
    set $result$ = getOutputStreamFromSection ("dosinanicon_start_myservice")
endif

[sub_find_myservice]
set $splitS$ = takeString (1, splitStringOnWhiteSpace("%s%"))
Set $compareS$ = $splitS$ + takeString(1, splitString("%s%", $splitS$))
if $compareS$ = $myservice$
    set $found$ = "true"
endif
```

```
[dosinainicon_start_mysevice]
net start "$myservice$"

[dosbatch_netcall]
@echo off
net start
```

12.3 Skript für Installationen im Kontext eines lokalen Administrators

In manchen Situationen kann es sinnvoll oder notwendig sein, ein *opsi-winst/opsi-script* Skript als lokal eingeloggtter Benutzer auszuführen anstatt wie üblich im Kontext eines Systemdienstes. Beispielsweise kann es sein, dass Softwareinstallationen, die vom *opsi-winst/opsi-script* aus aufgerufen werden, zwingend einen Benutzerkontext benötigen oder dass bestimmte Dienste, die für den Installationsvorgang wichtig sind, erst nach dem Login zur Verfügung stehen.

MSI-Installationen, die einen lokalen User benötigen lassen, sich häufig durch die Option *ALLUSERS=1* dazu "überreden" auch ohne aus zukommen. Beispiel:

```
[Actions]
DefVar $LOG_LOCATION$
Set $LOG_LOCATION$ = %opsiLogDir% + "\myproduct.log"
winbatch_install_myproduct

[winbatch_install_myproduct]
msiexec /qb ALLUSERS=1 /l* $LOG_LOCATION$ /i %SCRIPTPATH%\files\myproduct.msi
```

Eine andere aufwendigere Möglichkeit dieses Problem zu lösen, ist einen administrativen User temporär anzulegen und diesen zur Installation des Programms zu verwenden.

Dazu gehen Sie wie folgt vor:

Erzeugen Sie ein neues Produkt auf Basis des Produkts *opsi-template-with-admin*. Legen im Verzeichnis *install\productid* ein Verzeichnis *localsetup* an. Verschieben Sie die gesamten Installationsdateien Ihres Produktes in das Unterverzeichnis *localsetup* des erzeugten Produktes.

Sorgen Sie dafür, dass Ihr setup-script einen **Reboot** auslöst. Dazu fügen Sie am besten direkt unter **[Actions]** die Zeile **ExitWindows /Reboot** ein:

```
[Actions]
ExitWindows /Reboot
```

Das nachfolgende abgedruckte *opsi-winst/opsi-script* Skript-Template erzeugt temporär den gewünschten Benutzerkontext, führt in ihm eine Installation aus und beseitigt ihn schließlich wieder. Für die Verwendung sind die folgende Variablen zu setzen:

- der richtige Wert für die Variable **\$Productname\$**,
- der richtige Wert für die Variable **\$ProductSize\$** und
- der richtige Wert für die Variable **\$LocalSetupScript\$** (Name des eigentlichen setup-scriptes)

Das Skript führt im Einzelnen folgende Schritte aus:

- Anlegen eines lokalen Administrator *opsiSetupAdmin*;
- Sichern des bisherigen Autologon-Zustands;
- Eintragen des *opsiSetupAdmin* als Autologon-User;

- Installationsdateien auf den Client kopieren (wohin steht in `$localFilePath$`), dort befindet sich das Installationskript, das als lokaler Benutzer ausgeführt werden soll;
- RunOnce-Eintrag in der Registry anlegen, der den `opsi-winst/opsi-script` mit dem lokalen Skript als Argument aufruft;
- Neustart des Client (damit die Änderungen an der Registry Wirkung haben);
- `opsi-winst/opsi-script` startet und führt `ExitWindows /ImmediateLogout` aus: durch den Autologon meldet sich nun automatisch der Benutzer `opsiSetupAdmin` an, es wird der RunOnce-Befehl ausgeführt;
- nun läuft die Installation ganz normal, jedoch am Ende des Skripts muss zwingend neugestartet werden (also mit `ExitWindows /ImmediateReboot`), da sonst die Oberfläche des momentan eingeloggten Users `opsiSetupAdmin` mit Administratorrechten(!) freigegeben wird;
- nach dem Reboot wird wieder aufgeräumt (alten Zustand von Autologon wiederherstellen, lokale Setup-Dateien löschen, Benutzerprofil von `opsiSetupAdmin` löschen).

Wie man sieht, gliedert sich die Installation in 2 Bereiche: ein Skript, das als Service ausgeführt wird, alles zum lokalen Login vorbereitet und später wieder aufräumt (Masterscript) und ein Skript, dass als lokaler Administrator ausgeführt wird und die eigentliche Setup-Routine für das Produkt enthält (Localscript).



Achtung

Erfordert das Localscript mehr als nur einen Reboot, muss auch das Masterscript verändert bzw. um diese Anzahl von Reboots erweitert werden. Solange das Localscript nicht fertig ist, muss das Masterscript ein `ExitWindows /ImmediateLogout` ausführen, um die Kontrolle an das Localscript zu übergeben. Der RunOnce-Eintrag muss dann immer wieder neu gesetzt werden. Ebenso müssen Username und Passwort des Autologins nach jedem Reboot neu gesetzt werden.

Es gibt (ab opsi 4.0.2-2) einen direkten Zugang vom lokalen Skript auf die Produkteigenschaften und das Lizenzmanagement mit den gewohnten Funktionen.

Es kann Produktinstallationen (also aus dem Localscript heraus) geben, die Schlüssel in der Registry verändern, die vorher vom Masterscript gesichert und am Ende durch dieses wieder überschrieben werden. In diesem Fall muss die Wiederherstellung der Werte im Masterscript unterbunden werden.

Das Localscript läuft unter eingeloggtem Administrator Account. Wenn hier nicht Keyboard/Maus gesperrt werden, besteht für den Anwender die Möglichkeit das Skript zu unterbrechen und Administrator zu werden.

Es gibt daher im Template ein Property `debug`. Steht dieses auf dem Default (=false) so werden Tastatur und Maus gesperrt und keine Passwörter geloggt. Während der Skriptentwicklung und den Tests kann dieses Property auf `true` gesetzt werden.

Das Passwort des temporären `opsiSetupAdmin` wird in nachfolgenden Beispiel durch die Funktion `RandomStr` bestimmt.



Wichtig

Verwenden Sie aktualisierte Versionen des folgenden Beispiels aus dem Templateprodukt: `opsi-template-with-admin`.

```
; Copyright (c) uib gmbh (www.uib.de)
; This sourcecode is owned by uib
; and published under the Terms of the General Public License.

; TEMPLATE for
; Skript fuer Installationen im Kontext eines temporaeren lokalen Administrators
; installations as temporary local admin
```

```

; see winst_manual.pdf / winst_handbuch.pdf

;
; !!! Das lokale Installations-Skript, das durch den temporaeren lokalen Admin ausgefuehrt wird
; !!! (sein Name steht in $LocalSetupScript$), muss mit dem Befehl
; !!! exitWindows /Reboot
; !!! enden
;
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
; Vorarbeiten/Voraussetzungen/Doku pruefen wie in Winsthandbuch
; Skript fuer Installationen im Kontext eines lokalen Administrators
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

[Actions]
requiredWinstVersion >= 4.11.4.12
setLogLevel=7
ScriptErrorMessages=off
DefVar $ProductName$
DefVar $ProductSizeMB$
DefVar $LocalSetupScript$
DefVar $LockKeyboard$
DefVar $OpsAdminPass$
DefVar $RebootFlag$
DefVar $WinstRegKey$
DefVar $AutoName$
DefVar $AutoPass$
DefVar $AutoDom$
DefVar $AutoLogon$
DefVar $AutoBackupKey$
DefVar $LocalFilesPath$
DefVar $LocalWinst$
DefVar $DefaultLogLevel$
DefVar $PasswdLogLevel$
DefVar $AdminGroup$
DefVar $SearchResult$
DefVar $LocalDomain$
DefVar $debug$
DefVar $isFatal$

; -----
; - Please edit the following values
; -----
Set $ProductName$ = "opsi-template-with-admin"
Set $ProductSizeMB$ = "1"
Set $LocalSetupScript$ = "setup32.opsiscript"
; -----

comment "get and set initial values..."
set $debug$ = GetProductProperty("debug","false")
set $isFatal$ = "false"
set $DefaultLogLevel$ = "7"
SetLogLevel=$DefaultLogLevel$
Set $LocalDomain$ = "%PCNAME%"

comment "check if we productive or debugging..."

```

```

if $debug$ = "true"
    comment "we are in debug mode"
    Set $LockKeyboard$="false"
    Set $PasswdLogLevel$="7"
else
    comment "we are in productive mode"
    comment "set $LockKeyboard$ to true to prevent user hacks while admin is logged in"
    Set $LockKeyboard$="true"
    comment " set $PasswdLogLevel$ to 0 for production"
    Set $PasswdLogLevel$="0"
endif

comment "handle Rebootflag"
Set $WinstRegKey$ = "HKLM\SOFTWARE\opsi.org\winst\"+$ProductName$
Set $RebootFlag$ = GetRegistryStringValue32(["+$WinstRegKey$+" "RebootFlag")

comment "some paths required"
Set $AutoBackupKey$ = $WinstRegKey$\AutoLogonBackup"
Set $LocalFilePath$ = "C:\opsi.org\tmp\opsi_local_inst"
Set $LocalWinst$ = "%ProgramFilesDir%\opsi.org\opsi-client-agent\opsi-winst\winst32.exe"
if not( FileExists($LocalWinst$) )
    LogError "No opsi-winst found. Aborting."
    isFatalError
endif

comment "show product picture"
ShowBitmap "%scriptpath%\localsetup\"+$ProductName$.png" $ProductName$

if not (($RebootFlag$ = "1") or ($RebootFlag$ = "2") or ($RebootFlag$ = "3"))
    comment "Part before first Reboot"
    comment "just reboot - this must be done if this is the first product after OS
    installation"
    comment "handle Rebootflag"
    Set $RebootFlag$ = "1"
    Registry_SaveRebootFlag /32bit
    ;ExitWindows /ImmediateReboot
endif ; Rebootflag = not (1 or 2 or 3)

if $RebootFlag$ = "1"
    comment "Part before second Reboot"
    setActionProgress "Preparing"

    if not(HasMinimumSpace ("%SYSTEMDRIVE%", ""+$ProductSizeMB$+" MB"))
        LogError "Not enough space on drive C: . "+$ProductSizeMB$+" MB on C: required
        for "+$ProductName$
        isFatalError
    endif

    comment "Lets work..."
    Message "Preparing "+$ProductName$+" install step 1..."
    sub_Prepare_AutoLogon

    comment "we need to reboot now to be sure that the autologon work"
    comment "handle Rebootflag"
    Set $RebootFlag$ = "2"
    Registry_SaveRebootFlag /32bit
    ExitWindows /ImmediateReboot

```

```

endif ; Rebootflag = not (1 or 2)

if ($RebootFlag$ = "2")
    comment "Part after first Reboot"

    comment "handle Rebootflag"
    Set $RebootFlag$ = "3"
    Registry_SaveRebootFlag /32bit

    comment "Lets work..."
    Message "Preparing "+$ProductName$+" install step 2..."
    Registry_enable_keyboard /sysnative

    comment "now let the autologon work"
    comment "it will stop with a reboot"
    setActionProgress "Run Installation"

    ExitWindows /ImmediateLogout
endif ; Rebootflag = 2

if ($RebootFlag$ = "3")
    comment "Part after second Reboot"
    ExitWindows /Reboot
    setActionProgress "Cleanup"
    comment "handle Rebootflag"
    Set $RebootFlag$ = "0"
    Registry_SaveRebootFlag /32bit

    comment "Lets work..."
    Message "Cleanup "+$ProductName$+" install (step 3)..."
    sub_Restore_AutoLogon
    set $SearchResult$ = GetRegistryStringValueSysnative(" [HKLM\SOFTWARE\Microsoft\Windows\
CurrentVersion\RunOnce] opsi_autologon_setup")
    if $SearchResult$ = $LocalWinst$+" "+$LocalFilePath$+"\ "+$LocalSetupScript$+" /batch /
productid %installingProdName%"
        LogError "Localscript did not run. We remove the RunOnce entry and abort"
        Registry_del_runonce /sysnative
        set $isFatal$ = "true"
    endif
    if "true" = getRegistryStringValue32(" [HKLM\Software\opsi.org\winst] with-admin-fatal")
        LogError "set to fatal because the local script stored this result"
        set $isFatal$ = "true"
    endif
    comment "cleanup the registry key which stores a fatal result of the local script"
    Registry_clean_fatal_flag /32bit
    if $isFatal$ = "true"
        isFatalError
    endif
    comment "This is the clean end of the installation"
endif ; Rebootflag = 3

[sub_Prepare_AutoLogon]
comment "copy the setup script and files"
Files_copy_Setup_files_local
comment "read actual Autologon values for backup"
set $AutoName$ = GetRegistryStringValueSysnative(" [HKLM\SOFTWARE\Microsoft\Windows NT\

```

```

    CurrentVersion\Winlogon] DefaultUserName")
comment "if AutoLogonName is our setup admin user, something bad happend"
comment "then let us cleanup"
if ($AutoName$="opsiSetupAdmin")
    set $AutoName$=""
    set $AutoPass$=""
    set $AutoDom$=""
    set $AutoLogon$="0"
else
    set $AutoPass$ = GetRegistryStringValueSysnative(" [HKLM\SOFTWARE\Microsoft\Windows NT\
    CurrentVersion\Winlogon] DefaultPassword")
    set $AutoDom$ = GetRegistryStringValueSysnative(" [HKLM\SOFTWARE\Microsoft\Windows NT\
    CurrentVersion\Winlogon] DefaultDomainName")
    set $AutoLogon$ = GetRegistryStringValueSysnative(" [HKLM\SOFTWARE\Microsoft\Windows NT\
    CurrentVersion\Winlogon] AutoAdminLogon")
endif

comment "backup AutoLogon values"
Registry_save_autologon /32bit

comment "prepare the admin AutoLogon"
SetLogLevel=$PasswdLogLevel$
set $OpsAdminPass$= randomstr
Registry_autologon /sysnative

comment "get the name of the admin group"
set $AdminGroup$ = SidToName("S-1-5-32-544")
comment "create our setup admin user"
DosInAnIcon_makeadmin
SetLogLevel=$DefaultLogLevel$

comment "store our setup script as run once"
Registry_runOnce /sysnative

comment "disable keyboard and mouse while the autologin admin works"
if ($LockKeyboard$="true")
    Registry_disable_keyboard /Sysnative
endif

comment "cleanup the registry key which stores a fatal result of the local script"
Registry_clean_fatal_flag /32bit

[sub_Restore_AutoLogon]
comment "read AutoLogon values from backup"
set $AutoName$ = GetRegistryStringValue("["+ $AutoBackupKey$+ "] DefaultUserName")
set $AutoPass$ = GetRegistryStringValue("["+ $AutoBackupKey$+ "] DefaultPassword")
set $AutoDom$ = GetRegistryStringValue("["+ $AutoBackupKey$+ "] DefaultDomainName")
set $AutoLogon$ = GetRegistryStringValue("["+ $AutoBackupKey$+ "] AutoAdminLogon")

comment "restore the values"
SetLogLevel = $PasswdLogLevel$
Registry_restore_autologon /Sysnative
SetLogLevel = $DefaultLogLevel$
comment "delete our setup admin user"
DosInAnIcon_deleteadmin
comment "cleanup setup script, files and profiledir"
Files_delete_Setup_files_local

```

```

comment "delete profiledir"
DosInAnIcon_deleteprofile

[Registry_save_autologon]
openkey [$AutoBackupKey$]
set "DefaultUserName"="$AutoName$"
set "DefaultPassword"="$AutoPass$"
set "DefaultDomainName"="$AutoDom$"
set "AutoAdminLogon"="$AutoLogon$"

[Registry_restore_autologon]
openkey [HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon]
set "DefaultUserName"="$AutoName$"
set "DefaultPassword"="$AutoPass$"
set "DefaultDomainName"="$AutoDom$"
set "AutoAdminLogon"="$AutoLogon$"

[DosInAnIcon_deleteadmin]
NET USER opsiSetupAdmin /DELETE

[Registry_SaveRebootFlag]
openKey [$WinstRegKey$]
set "RebootFlag" = "$RebootFlag$"

[Files_copy_Setup_files_local]
copy -s "%ScriptPath%\localsetup\*.*" "$LocalFilesPath$"

[Files_delete_Setup_files_local]
del -sf $LocalFilesPath\
; folgender Befehl funktioniert nicht vollständig, deshalb ist er zur Zeit auskommentier
; der Befehl wird durch die Sektion "DosInAnIcon_deleteprofile" ersetzt (P.Ohler)
;delete -sf "%ProfileDir%\opsiSetupAdmin"

[DosInAnIcon_deleteprofile]
rmdir /S /Q "%ProfileDir%\opsiSetupAdmin"

[DosInAnIcon_makeadmin]
NET USER opsiSetupAdmin $OpsAdminPass$ /ADD
NET LOCALGROUP $AdminGroup$ /ADD opsiSetupAdmin

[Registry_autologon]
openkey [HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon]
set "DefaultUserName"="opsiSetupAdmin"
set "DefaultPassword"="$OpsAdminPass$"
set "DefaultDomainName"="$LocalDomain$"
set "AutoAdminLogon"="1"

[Registry_runonce]
openkey [HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce]
set "opsi_autologon_setup"='"$LocalWinst$" "$LocalFilesPath$\$LocalSetupScript$" /batch /
productid %installingProdName%'

[Registry_del_runonce]
openkey [HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce]
DeleteVar "opsi_autologon_setup"

[Registry_disable_keyboard]

```

```

openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kbdclass]
set "Start"=REG_DWORD:0x4
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Mouclass]
set "Start"=REG_DWORD:0x4

[Registry_enable_keyboard]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kbdclass]
set "Start"=REG_DWORD:0x1
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Mouclass]
set "Start"=REG_DWORD:0x1

[Registry_clean_fatal_flag]
openkey [WinstRegKey$]
DeleteVar "with-admin-fatal"

```

12.4 XML-Datei patchen: Setzen des Vorlagenpfades für OpenOffice.org 2.0

Das Setzen des Vorlagenpfades kann mit Hilfe der folgenden Skriptteile erfolgen:

```

[Actions]
; ....

DefVar $oooTemplateDirectory$
;-----
;set path here:

Set $oooTemplateDirectory$ = "file://server/share/verzeichnis"
;-----
;...

DefVar $sofficePath$
Set $sofficePath$= GetRegistryStringValue (" [HKEY_LOCAL_MACHINE\SOFTWARE\OpenOffice.org\
OpenOffice.org\2.0] Path")
DefVar $oooDirectory$
Set $oooDirectory$= SubstringBefore ($sofficePath$, "\program\soffice.exe")
DefVar $oooShareDirectory$
Set $oooShareDirectory$ = $oooDirectory$ + "\share"

XMLPatch_paths_xcu $oooShareDirectory$+"\registry\data\org\openoffice\Office\Paths.xcu"
; ...

[XMLPatch_paths_xcu]
OpenNodeSet
- error_when_no_node_existing false
- warning_when_no_node_existing true
- error_when_nodecount_greater_1 false
- warning_when_nodecount_greater_1 true
- create_when_node_not_existing true
- attributes_strict false

documentroot
all_childelements_with:
elementname: "node"
attribute:"oor:name" value="Paths"

```

```

all_childelements_with:
elementname: "node"
attribute: "oor:name" value="Template"
all_childelements_with:
elementname: "node"
attribute: "oor:name" value="InternalPaths"
all_childelements_with:
elementname: "node"

end

SetAttribute "oor:name" value="$oooTemplateDirectory$"

```

12.5 XML-Konfiguration für eine MsSql-Anwendung patchen: Ein Beispiel mit irreführend benannten Attributen

Die Ausgangsdatei für den Patch hat z.B: folgende Form, DataSource und InitialCatalog sollen dynamisch gesetzt werden mit Hilfe der Variablen \$source\$ und \$catalog\$.

```

<?xml version=" 1.0 " ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <appSettings>
    <add key="Database.DatabaseType" value="MsSqlServer" />
    <add key="Database.DataSource" value="[db-servername] \ [db-instance]" />
    <add key="Database.InitialCatalog" value="TrustedData" />
    <add key="ActiveDirectory.Enabled" value="false" />
    <add key="ActiveDirectory.LdapRoot" value="" />
  </appSettings>
</configuration>

```

Dann kann man mit folgender XMLPatch-Sektion arbeiten:

```

[XMLPatch_db_config]
openNodeSet
  documentroot
  all_childelements_with:
    elementname: "appSettings"
  all_childelements_with:
    elementname: "add"
    attribute: "key" value = "Database.DataSource"
end
SetAttribute "value" value="$source$"

openNodeSet
  documentroot
  all_childelements_with:
    elementname: "appSettings"
  all_childelements_with:
    elementname: "add"
    attribute: "key" value = "Database.InitialCatalog"
end
SetAttribute "value" value="$catalog$"

```

12.6 XML-Datei einlesen mit dem opsi-winst

Wie bereits im vorangehenden [Kapitel "XML-Datei patchen"](#) beschrieben, lassen sich auch XML-Dateien mit dem *opsi-winst/opsi-script* einlesen. Hier soll nun exemplarisch gezeigt werden, wie man die Werte eines bestimmten Knotens ausliest. Als Quelle dient dazu folgende XML-Datei:

```
<?xml version="1.0" encoding="utf-16" ?>
<Collector xmlns="http://schemas.microsoft.com/appx/2004/04/Collector" xmlns:xs="http://www.w3.
  org/2001/XMLSchema-instance" xs:schemaLocation="Collector.xsd" UtcDate="04/06/2006 12:28:17"
  LogId="{693B0A32-76A2-4FA0-979C-611DEE852C2C}" Version="4.1.3790.1641" >
  <Options>
    <Department></Department>
    <IniPath></IniPath>
    <CustomValues>
    </CustomValues>
  </Options>
  <SystemList>
    <ChassisInfo Vendor="Chassis Manufacture" AssetTag="System Enclosure 0" SerialNumber="EVAL"
    />
    <DirectxInfo Major="9" Minor="0"/>
  </SystemList>
  <SoftwareList>
    <Application Name="Windows XP-Hotfix - KB873333" ComponentType="Hotfix" EvidenceId="256"
    RootDirPath="C:\WINDOWS\NtUninstallKB873333$\spuninst" OsComponent="true" Vendor="Microsoft
    Corporation" Crc32="0x4235b909">
      <Evidence>
        <AddRemoveProgram DisplayName="Windows XP-Hotfix - KB873333" CompanyName="Microsoft
        Corporation" Path="C:\WINDOWS\NtUninstallKB873333$\spuninst" RegistryPath="
        HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KB873333"
        UninstallString="C:\WINDOWS\NtUninstallKB873333$\spuninst\spuninst.exe" OsComponent="true"
        UniqueId="256"/>
      </Evidence>
    </Application>
    <Application Name="Windows XP-Hotfix - KB873339" ComponentType="Hotfix" EvidenceId="257"
    RootDirPath="C:\WINDOWS\NtUninstallKB873339$\spuninst" OsComponent="true" Vendor="Microsoft
    Corporation" Crc32="0x9c550c9c">
      <Evidence>
        <AddRemoveProgram DisplayName="Windows XP-Hotfix - KB873339" CompanyName="Microsoft
        Corporation" Path="C:\WINDOWS\NtUninstallKB873339$\spuninst" RegistryPath="
        HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KB873339"
        UninstallString="C:\WINDOWS\NtUninstallKB873339$\spuninst\spuninst.exe" OsComponent="true"
        UniqueId="257"/>
      </Evidence>
    </Application>
  </SoftwareList>
</Collector>
```

Möchte man nur die Elemente und deren Werte aller „Application“-Knoten auslesen, kann man dies mit folgendem Code bewerkstelligen (nur Ausschnitt):

```
[Actions]
DefStringList $list$

...

set $list$ = getReturnListFromSection ('XMLPatch_findProducts '+$TEMP$+'\test.xml')
for $line$ in $list$ do Sub_doSomething
```

```

[XMLPatch_findProducts]
openNodeSet
    ; Knoten „Collector“ ist der documentroot
    documentroot
    all_childelements_with:
        elementname:"SoftwareList"
    all_childelements_with:
        elementname:"Application"
end
return elements

[Sub_doSomething]
set $escLine$ = EscapeString:$line$
; hier kann man nun diese Elemente in $escLine$ bearbeiten

```

Hier sieht man auch eine weitere Besonderheit. Es sollte vor dem Benutzen der eingelesenen Zeilen erst ein EscapeString der Zeile erzeugt werden, damit enthaltene Sonderzeichen nicht vom *opsi-winst/opsi-script* interpretiert werden. Die Zeile wird nun gekapselt behandelt, sonst könnten reservierte Zeichen wie \$,%," oder \ leicht zu unvorhersehbaren Fehlfunktionen führen.

,

12.7 Einfügen einer Namensraumdefinition in eine XML-Datei

Die *opsi-winst/opsi-script* XMLPatch-Sektion braucht eine voll ausgewiesenen XML Namensraum (wie es im XML RFC gefordert wird). Aber es gibt XML Konfigurationsdateien, in denen „nahe liegende“ Elemente nicht deklariert werden (und auslesende Programme, die auch davon ausgehen, dass die Konfigurationsdatei entsprechend aussieht).

Besonders das Patchen der meisten XML/XCU Konfigurationsdateien von OpenOffice.org erweist sich als sehr schwierig. Um dieses Problem zu lösen hat A. Pohl (Vielen Dank!) die Funktionen XMLAddNamespace und XMLremoveNamespace entwickelt. Die Funktionsweise ist im folgenden Beispiel demonstriert:

```

DefVar $XMLFile$
DefVar $XMLElement$
DefVar $XMLNameSpace$
set $XMLFile$ = "D:\Entwicklung\OPSI\winst\Common.xcu3"
set $XMLElement$ = 'oor:component-data'
set $XMLNameSpace$ = 'xmlns:xml="http://www.w3.org/XML/1998/namespace"'

if XMLAddNamespace($XMLFile,$XMLElement$, $XMLNameSpace$)
    set $NSMustRemove$="1"
endif
;
; now the XML Patch should work
; (commented out since not integrated in this example)
;
; XMLPatch_Common $XMLFile$
;
; when finished we rebuild the original format
if $NSMustRemove$="1"
    if not (XMLRemoveNamespace($XMLFile,$XMLElement,$XMLNameSpace$))
        LogError "XML-Datei konnte nicht korrekt wiederhergestellt werden"
        isFatalError
    endif
endif
endif

```

Es ist zu beachten, dass die XML Datei so formatiert wird, dass der Element-Tag-Bereich keine Zeilenumbrüche enthält.

12.8 Herausfinden, ob ein Skript im Kontext eines bestimmten Events läuft

Der *opsiclientd* bestimmt und weiß, welches Event gerade aktiv ist. *opsi-script* kann sich mittels eines *opsiservicecall* mit dem *opsiclientd* verbinden und Events abfragen:

```
[actions]
setLogLevel=5
DefVar $queryEvent$
DefVar $result$

;=====
set $queryEvent$ = "gui_startup"

set serviceInfo = getReturnListFromSection('opsiservicecall_event_on_demand_is_running /
opsiclientd')
set $result$ = takestring(0, serviceInfo)
if $result$ = "true"
    comment "event " + $queryEvent$ + " is running"
else
    comment "NOT running event " + $queryEvent$
endif

;=====
set $queryEvent$ = "on_demand"

set serviceInfo = getReturnListFromSection('opsiservicecall_event_on_demand_is_running /
opsiclientd')
set $result$ = takestring(0, serviceInfo)
if $result$ = "true"
    comment "event " + $queryEvent$ + " is running"
else
    comment "NOT running event " + $queryEvent$
endif

;=====
set $queryEvent$ = "on_demand{user_logged_in}"

set serviceInfo = getReturnListFromSection('opsiservicecall_event_on_demand_is_running /
opsiclientd')
set $result$ = takestring(0, serviceInfo)
if $result$ = "true"
    comment "event " + $queryEvent$ + " is running"
else
    comment "NOT running event " + $queryEvent$
endif
```

Kapitel 13

Spezielle Fehlermeldungen

- Keine Verbindung mit dem opsi-Service
Der *opsi-winst/opsi-script* meldet "... cannot connect to service".

Hinweise auf mögliche Probleme gibt die dazu angezeigte Nachricht:

Socket-Fehler #10061, Verbindung abgelehnt

Möglicherweise läuft der Service nicht

Socket-Fehler #10065, Keine Route zum Host

Keine Netzwerkverbindung zum Server

HTTP/1.1. 401 Unauthorized

Der Service antwortet, akzeptiert aber das Passwort nicht.

Kapitel 14

opsi documentation generator

14.1 Introduction

The opsi-doc-generator is a program to create documentations in asciidoc format from the following sources:

- opsi-script library files (opsi-script)
- opsi webservice interface definition files (python) (implementation at work)

Why asciidoc as output ?

- asciidoc is the standard format for all opsi documentatation
- asciidoc is a base format to create documents in different formats like `html`, `pdf`, `epub`, `docbook`, ...

The documentation is created from informations, that are extracted from the source code. From the source code opsi-doc-generator knows the definitions of opsi-script defined functions and can get the informazion from there. For additional information there may be special markers in comment lines which hold additional information on the level of file, function and parameter.

14.2 opsi-doc-generator program

There are two variants of this program:

- The GUI Version `opsi-doc-generator-gui`:
- The CLI Version `opsi-doc-generator`:

```
$ ./opsi_doc_generator --help
Creates asciidoc from commented opsi-script library code
and calls asciidoctor to convert asciidoc to html
and shows created html file in browser.
opsi_doc_generator
Version: 4.1.0.0
Usage:
opsi_doc_generator [Options] inputfile
Options:
--help -> write this help and exit
```

You will find this programs for Linux and Windows as opsi packages in the contribute area on download.uib.de

14.3 opsi-doc-generator marker

There are three different levels where information can be found in a source file:

- file
- function (may be more than one in a file)
- function parameter (may be more than one in a function)

Every marker starts with the language specific comment char (opsiscript=;) followed by the the @ char and the marker identifier string.

Every allowed marker can occur never, once or multiple time on a level. If a marker occur multiple times, all lines of this marker are concatenated

After a marker one or more space chars have to be used before the information start

OPSI-SCRIPT MARKERS ALLOWED ON THE FILE LEVEL

- ;@author
- ;@email
- ;@date
- ;@copyright
- ;@version
- ;@filedesc Description of file

OPSI-SCRIPT MARKERS ALLOWED ON THE FUNCTION LEVEL

- ;@author Author (if absent author of file is used)
- ;@email eMail address (if absent eMail of file is used)
- ;@date Date (if absent date of file is used)
- ;@copyright copyright (if absent copyright of file is used)
- ;@version version (if absent version of file is used)
- ;@Description Description of function
- ;@Returns Return value of function
- ;@OnError What happens in the case of an error
- ;@SpecialCase What happens in known special unexpected cases like empty input, no network, and so son
- ;@Requires
- ;@References The name of an other function in this file that are related to this function. only one per line. For multiple references use multiple lines
- ;@Links
- ;@Example An example for the use of this function.
Examples are in most cases multiline with idents. The start of the information in the first example line defines the base ident. Idents have to be done with space chars only (no tabs).

OPSI-SCRIPT MARKERS ALLOWED ON THE FUNCTION PARAMETER LEVEL

- ;@ParamDesc_<pram name> Description of the parameter <pram name>
- ;@ParamAdvice_<pram name> Advice to the parameter <pram name>. That may be for example restrictions for valid values.

14.4 opsi-doc-generator examples

opsi-script markers on the file level

```
;@author      detlef oertel
;@email       d.oertel@uib.de
;@date        17.4.2018
;@copyright   GPLv3
;@version     1.0
;@filedesc   Collection of functions that manipulate the opsi backend via opsi service call
```

opsi-script markers on the function and parameter level

```
;@author      detlef oertel
;@date        17.5.2018
;@Description  Sets for the given list of opsi productIds the action request
;@Description  to 'setup' (also resolving the dependencies)
;@Returns     Returns string "true" if all is ok
;@OnError     Returns string "false"
;@SpecialCase Works only in opsi service mode (not in interactive or batch mode)
;@References
;@Links
;@ParamDesc_ $productlist$      List of opsi product Ids
;@ParamAdvice_ $productlist$
;@Example     [actions]
;@Example     DefStringlist $productlist$
;@Example     set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
;@Example     if not(stringtobool(setProductsToSetup($productlist$)))
;@Example     comment "call of setProductsToSetup failed"
;@Example     endif
```

Kapitel 15

opsi-winst Übungen

15.1 Einführung

Dieses Tutorial soll Funktionen der Winst-Scripte durch einfache Übungen aufzeigen. Wenn möglich sollen Konstanten und Variablen verwendet werden, wie zB. `%system%` für den Aufruf `c:\windows\system32`.

Zur Erstellung der Scripte sei auf folgende Dokumentationen verwiesen:

- opsi-winst Handbuch
- opsi-winst reference card

Außerdem gibt es unter download.uib.de im Bereich Testing oder Stable das Produkt opsi-script-test<Version>, indem ein setup.ins mit nahezu allen Winst-Befehlen enthalten ist.

15.2 Erstellung von Winst-Scripten

Die winst-Scripte lassen sich mit einem normalen Texteditor erstellen, deutlich übersichtlicher wird es jedoch wenn der verwendete Editor über ein Syntax-Highlighting verfügt. Die Scripte können aus einem manuel aufgerufenen Winst gestartet werden.

15.2.1 1. Übung

In der ersten Übung sollen alle Dateien aus dem Verzeichnis `c:\windows\system32` aufgelistet werden.

Verwenden Sie hierzu die Funktion "DosInAnIcon" mit der Sie Befehle wie in einer Dos-Box absetzen können.

Folgende opsi-winst Funktionen können Sie verwenden:

- `DosInAnIcon`

15.2.2 2. Übung

Erweitern Sie das Script aus Übung 1, indem sie die Liste aller Dateien aus dem System32-Verzeichnis in eine Stringliste übergeben.

Zur Erstellung einer solchen Liste kann der Funktion "getOutputStreamFromSection" der Aufruf von "DosInAnIcon" übergeben werden.

Folgende opsi-winst Funktionen können Sie verwenden:

- `DefStringlist`
- `getOutputStreamFromSection`
- `setloglevel = 7`

15.2.3 3. Übung

In einer Kommentarzeile soll die Anzahl der DDL-Files unter `c:\windows\system32` ausgegeben werden.

Erweitern Sie das Script aus Übung 2, indem Sie aus der Liste der Dateien mit Hilfe der Funktionen "getListContaining" die DDL-Files separieren. Mit Hilfe von "count" können Sie die Anzahl auslesen.

Folgende opsi-winst Funktionen können Sie verwenden:

- `getListContaining`
- `count`
- `comment`

15.2.4 4. Übung

Gibt es eine `kernel32.dll` in `c:\windows\system32` und wenn ja welche Größe hat sie?

Aufbauend auf dem Script aus Übung 3 können Sie mit Hilfe der Funktionen "TakeFirstStringContaining" den Eintrag mit `Kernel32.dll` gewinnen. Denken Sie dabei daran, dass das Ergebnis keine Liste ist. Mit der Funktion "SplitStringOnWhiteSpace" läßt sich die Ausgabe in eine String-Liste überführen, aus der mit Hilfe von "TakeString" die Größe der Datei ausgelesen werden kann.

Folgende opsi-winst Funktionen können Sie verwenden:

- `TakeFirstStringContaining`
- `SplitStringOnWhiteSpace`
- `TakeString`

15.2.5 5. Übung

Welche `kernel32.dll` auf einem 64Bit-System ist größer, die 32Bit- oder die 64Bit-Version?

Erweitern Sie das Script aus Übung 4 entsprechend. Die 64Bit-Version der `kernel32.dll` erhalten Sie, indem die Dos-Box in der 64Bit-Version aufgerufen wird.

Folgende opsi-winst Funktionen können Sie verwenden:

- `DosInAnIcon winst /64bit`

15.3 Lösungen

15.3.1 Lösung Übung 1

```
[Actions]
comment "Show all Systemfiles"
DosInAnIcon_Dir

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir
```

15.3.2 Lösung Übung 2

```
[Actions]
setloglevel = 7

DefStringList $list1$

comment "Show all Systemfiles"
;DosInAnIcon_Dir
comment "Output from DosInAnIcon is setting to a list"
set $list1$ = getOutputStreamFromSection ("DosInAnIcon_Dir")

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir
```

15.3.3 Lösung Übung 3

```
[Actions]
DefVar $DLLAnzahl$
DefStringList $list1$

comment "Show all Systemfiles"
;DosInAnIcon_Dir
comment "Output from DosInAnIcon is setting to a list"
set $list1$ = getOutputStreamFromSection ("DosInAnIcon_Dir")
;getListContaining(<list>,<search string>)
;get a partial list with all strings that match <search string>
comment "list with only DDL-Files"
set $list1$ = getListContaining ($list1$,".dll")
comment "Number of DDL-Files"
set $DLLAnzahl$ = count ($list1$)
comment "Number of DLL-Files: " + $DLLAnzahl$

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir *.*
```

15.3.4 Lösung Übung 4

```
[Actions]
DefVar $DLLAnzahl$
DefVar $dirline$

DefStringList $list1$

comment "Show all Systemfiles"
;DosInAnIcon_Dir
comment "Output from DosInAnIcon is setting to a list"
set $list1$ = getOutputStreamFromSection ("DosInAnIcon_Dir")
;set $list64$ = getOutputStreamFromSection ("DosInAnIcon_Dir winst /64bit")
comment "get string kernel32.dll"
set $dirline$ = takeFirstStringContaining ($list1$,"kernel32.dll")
if $dirline$ = ""
comment "Kernel32.dll not exist"
else
    set $list1$ = splitStringOnWhiteSpace($dirline$)
    set $dirline$ = takeString (2,$list1$)
    comment "Size of Kernel32.dll: "+$dirline$+" B"
endif

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir *.*
```

15.3.5 Lösung Übung 5

```
[Actions]
DefVar $dirline$
DefVar $dirline64$
DefStringList $list32$
DefStringList $list64$

;search for 32 Bit-Version
comment "Output from DosInAnIcon is setting to a list"
set $list32$ = getOutputStreamFromSection ("DosInAnIcon_Dir")
    comment "get string kernel32.dll"
set $dirline$ = takeFirstStringContaining ($list32$,"kernel32.dll")
if $dirline$ = ""
    comment "Kernel32.dll not exist"
else
    set $list32$ = splitStringOnWhiteSpace($dirline$)
    set $dirline$ = takeString (2,$list32$)
    comment "Size of 32Bit Kernel32.dll: "+$dirline$+" B"
endif

;search for 64 Bit-Version
set $list64$ = getOutputStreamFromSection ("DosInAnIcon_Dir winst /64bit")
comment "get string kernel32.dll"
set $dirline64$ = takeFirstStringContaining ($list64$,"kernel32.dll")
if $dirline64$ = ""
    comment "Kernel32.dll not exist"
```

```
else
    set $list64$ = splitStringOnWhiteSpace($dirline64$)
    set $dirline64$ = takeString (2,$list64$)
    comment "Size of 64 Bit Kernel32.dll: "+$dirline64$+" B"
endif

if $dirline64$ > $dirline$
    Comment "The 64Bit-Version is " +$dirline64$+ " Byte is greater than the 32Bit-Version
    with " +$dirline$+" Byte"
else
    Comment "The 32Bit-Version ist " +$dirline$+ " Byte is greater than the 64Bit-Version
    with " +$dirline64$+" Byte"
endif

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir *.*
```

Kapitel 16

opsi-script libraries

16.1 opsi-script libraries von uib

Dokumentation der opsi-script libraries welche Bestandteil des opsi-script / opsi-winst sind und von der uib gmbh gepflegt werden.

16.1.1 Documentation of opsi library: uib_backend.opsiscript

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

16.1.1.1 Documentation of local function setProductsToSetup

Definition

```
setProductsToSetup($productlist$ : stringlist) : string
```

Description

Sets for the given list of opsi productIds the action request to *setup* (also resolving the dependencies)

- Parameter: `$productlist$`
 - Type: `Stringlist` - Calltype: `CallByValue`
 - Parameter `$productlist$` Description:
List of opsi product Ids
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
[actions]
DefStringlist $productlist$

set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
if not(stringtobool(setProductsToSetup($productlist$)))
    comment "call of setProductsToSetup failed"
endif
```

16.1.1.2 Documentation of local function getInstalledLocalbootProducts

Definition

getInstalledLocalbootProducts(ref \$productlist\$: stringlist) : string

Description

Gets a list of productIds which are

- known to the client (productOnClient object exists)
- and localboot products. to *setup* (also resolving the dependencies)
- Parameter: \$productlist\$
 - Type: Stringlist - Calltype: CallByReference
 - Parameter \$productlist\$ Description: Output list of opsi product Ids that were be found
 - Parameter \$productlist\$ Advice: May be empty
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 20.4.2018
- Email: d.oertel@uib.de
- Version: 1.1
- Copyright: AGPLv3

Example:

```
if stringToBool(getInstalledLocalbootProducts($resultlist$))
    comment "getInstalledLocalbootProducts successful finished"
else
    LogError "getInstalledLocalbootProducts failed"
endif
set $tmplist$ = getListContainingList($baseproducts$, $resultlist$)
if stringToBool(compareLists($tmplist$, $baseproducts$))
    comment "check installed products successful finished"
else
    LogError "check installed products failed"
endif
comment " now install and rest products ...."
if stringToBool(setProductsToSetup($resetproducts$))
    comment "setProductsToSetup successful finished"
else
    LogError "setProductsToSetup failed"
endif
```

16.1.1.3 Documentation of local function delOpsiPoc

Definition

```
delOpsiPoc($donotdelList$ : stringlist) : string
```

Description

Delete all productOnClientObjects for this client and only for localboot products and not for products that are included in the \$donotdelList\$ parameter

- Parameter: \$donotdelList\$
 - Type: Stringlist - Calltype: CallByValue
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
set $opsiMetaDataFile$ = $targetDir+"\poc.json"
if Fileexists($opsiMetaDataFile$)
    comment "Delete existing meta data ...."
    set $tmplist$ = createStringList ("opsi-vhd-tester","opsi-vhd-control","opsi-vhd-auto-
upgrade")
    if stringToBool(delOpsiPoc($tmplist$))
        comment "Delete existing meta data successful finished"
    else
        LogError "Delete existing meta data failed"
    endif
    comment "Restore existing meta data ...."
    if stringToBool(restoreOpsiPoc($opsiMetaDataFile$))
        comment "Restore existing meta data successful finished"
    else
        LogError "Restore existing meta data failed"
    endif
else
    comment "No meta data existing - creating it...."
    if stringToBool(backupOpsiPoc($opsiMetaDataFile$))
        comment "Backup meta data successful finished"
    else
        LogError "Backup meta data failed"
    endif
endif
```

16.1.1.4 Documentation of local function backupOpsiPoc

Definition

```
backupOpsiPoc($filename$ : string) : string
```

Description

Get all localboot productOnClient objects for this client and write it to the json file \$filename\$

- Parameter: `$filename$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$filename$` Description:
Complete name of the file to create
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- References: [\[restoreOpsiPoc\]](#) [\[delOpsiPoc\]](#)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
see delOpsiPoc
```

16.1.1.5 Documentation of local function restoreOpsiPoc**Definition**

```
restoreOpsiPoc($filename$ : string) : string
```

Description

Load productOnClient objects from \$filename\$ and write it to to the server

- Parameter: `$filename$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$filename$` Description:
Complete name of the file to read
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
see delOpsiPoc
```

16.1.1.6 Documentation of local function `getInstalledLocalbootProductsWithVersion`

Definition

```
getInstalledLocalbootProductsWithVersion(ref $productlist$ : stringlist) : string
```

Description

Get all localboot `productOnClient` objects for this client and create a key/value list in the format `<productId>=<productVersion>-<packageVersion>` This list is be written to `$productlist$`

- Parameter: `$productlist$`
 - Type: `Stringlist` - Calltype: `CallByReference`
 - Parameter `$productlist$` Description:
The key/value list with all `<productId>=<productVersion>-<packageVersion>` of the client
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
comment "check for installed products ...."
if stringToBool(getInstalledLocalbootProductsWithVersion($installedproducts$))
    comment "getInstalledLocalbootProducts successful finished"
else
    LogError "getInstalledLocalbootProducts failed"
endif
comment "check for installable products ...."
if stringToBool(getInstallableLocalbootProductsWithVersion($possibleproducts$))
    comment "getInstallableLocalbootProductsWithVersion successful finished"
else
    LogError "getInstallableLocalbootProductsWithVersion failed"
endif
comment "fill $upgradeproducts$ .."
set $tmpplist$ = getKeyList($installedproducts$)
set $possibleproducts$ = getSubListByKey($tmpplist$, $possibleproducts$)
for %aktprod% in $installedproducts$ do sub_find_updatable_products
```

16.1.1.7 Documentation of local function `getInstallableLocalbootProductsWithVersion`

Definition

```
getInstallableLocalbootProductsWithVersion(ref $productlist$ : stringlist) : string
```

Description

Get all localboot `productOnDepot` objects for the depot of this client and create a key/value list in the format `<productId>=<productVersion>-<packageVersion>` This list is be written to `$productlist$`

- Parameter: `$productlist$`
 - Type: `Stringlist` - Calltype: `CallByReference`

- Parameter `$productlist$` Description:
The key/value list with all `<productId>=<productVersion>-<packageVersion>` of the depot of this client
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: GPLv3

Example:

```
see getInstalledLocalbootProductsWithVersion
```

16.1.2 Documentation of opsi library: uib_bootutils.opsiscript

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: GPLv3

16.1.2.1 Documentation of local function delFromWindowsBootmanager

Definition

```
delFromWindowsBootmanager($bootlabel$ : string) : string
```

Description

Deletes the boot entry given given by the parameter `$bootlabel$` from Windows boot manager by using `bcdedit`

- Parameter: `$bootlabel$`
 - Type: String - Calltype: CallByValue
 - Parameter `$bootlabel$` Description:
Windows boot manager entry label found in `bcdedit /v`
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [\[getWinBcdbootGuid\]](#) [\[bootNextToWinLabel\]](#) [\[bootNextToUefiLabel\]](#) [\[getUefiBcdbootGuid\]](#)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: GPLv3

Example:

```
[actions]
DefStringlist $productlist$

set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
if not(stringtobool(setProductsToSetup($productlist$)))
    comment "call of setProductsToSetup failed"
endif
```

16.1.2.2 Documentation of local function `getWinBcdbootGuid`

Definition

```
getWinBcdbootGuid($bootlabel$ : string) : string
```

Description

Get the boot entry GUID for the label given by the parameter `$bootlabel$` from Windows boot manager by using `bcdedit`

- Parameter: `$bootlabel$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$bootlabel$` Description: Windows boot manager entry label found in `bcdedit /v`
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [\[delFromWindowsBootmanager\]](#) [\[bootNextToWinLabel\]](#) [\[bootNextToUefiLabel\]](#) [\[getUefiBcdboot-Guid\]](#)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
Message "get windows boot guid ...."
set $windows_bcd_guid$ = getWinBcdbootGuid("WINDOWS.vhdx")
```

16.1.2.3 Documentation of local function `getUefiBcdbootGuid`

Definition

```
getUefiBcdbootGuid($bootlabel$ : string) : string
```

Description

Get the boot entry GUID for the label given by the parameter `$bootlabel$` from Windows boot manager by using `bcdedit`

- Parameter: `$bootlabel$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$bootlabel$` Description: UEFI boot manager entry label found in `bcdedit /enum firmware`
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [\[delFromWindowsBootmanager\]](#) [\[bootNextToWinLabel\]](#) [\[bootNextToUefiLabel\]](#) [\[getUefiBcdboot-Guid\]](#)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de

- Version: 1.0
- Copyright: AGPLv3

Example:

```
if runningonUefi
    set $peuefiguid$ = getUefiBcdbootGuid("opsitempwipe")
    set $exitcode$ = getlastexitcode
    if $exitcode$ = "0"
        if not ($peuefiguid$ = "")
            shellCall("bcdedit /delete "+$peuefiguid$)
        endif
    endif
endif
endif
```

16.1.2.4 Documentation of local function bootNextToWinLabel

Definition

```
bootNextToWinLabel($bootlabel$ : string) : string
```

Description

Sets the Windows bootmanager to boot next to the label given by the parameter \$bootlabel\$ from Windows boot manager by using bcdedit

- Parameter: \$bootlabel\$
 - Type: String - Calltype: CallByValue
 - Parameter \$bootlabel\$ Description: Windows boot manager entry label found in *bcdedit /v*
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [\[delFromWindowsBootmanager\]](#) [\[getUefiBcdbootGuid\]](#) [\[bootNextToUefiLabel\]](#)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
See bootNextToUefiLabel
```

16.1.2.5 Documentation of local function bootNextToUefiLabel

Definition

```
bootNextToUefiLabel($bootlabel$ : string) : string
```

Description

Sets the uefi bootmanager to boot next to the label given by the parameter \$bootlabel\$ from uefi boot manager by using bcdedit

- Parameter: `$bootlabel$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$bootlabel$` Description: UEFI boot manager entry label found in *bcdedit /enum firmware*
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [\[delFromWindowsBootmanager\]](#) [\[getUefiBcdbootGuid\]](#) [\[bootNextToUefiLabel\]](#)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
Message "Enable PE boot...."
if runningonuefi
    set $bootLabel$ = "opsitempwinpe"
    if not(stringToBool(bootNextToUefiLabel($bootLabel$)))
        logerror "Activating peboot is failed"
        isFatalError "failed peboot"
    endif
else
    set $bootLabel$ = "ramdisk=[boot]\sources\boot.wim"
    if not(stringToBool(bootNextToWinLabel($bootLabel$)))
        logerror "Activating peboot is failed"
        ;isFatalError "failed peboot"
    endif
endif
endif
```

16.1.2.6 Documentation of local function getDiskUuid

Definition

```
getDiskUuid($disknumber$ : string , $tmpdir$ : string ) : string
```

Description

Gets the disk uuid for the disk with the number `$disknumber$` by using diskpart. The temporary diskpart script is written to `$tmpdir$`

- Parameter: `$disknumber$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$disknumber$` Description: UEFI boot manager entry label found in *bcdedit /enum firmware*
- Parameter: `$tmpdir$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$tmpdir$` Description: Temporary directory to use
 - Parameter `$tmpdir$` Advice: Directory must exist
- Returns: Returns string "true" if all is ok

- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [\[enablePEPartition\]](#)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
for %disk% = "0" to calculate($diskcount1$+" -1") do set $aktdisklist$ = addtolist($aktdisklist$,
  getDiskUuid("%disk%", "x:")+"=%disk%")
set $disk$ = getvalue($diskuuid$, $aktdisklist$)
```

16.1.2.7 Documentation of local function enablePEPartition

Definition

```
enablePEPartition($disknumber$ : string , $partitionNumber$ : string, $pepartletter$ :
string, $useGpt$ : string) : string
```

Description

Try to make the partition \$partitionNumber\$ on the disk \$disknumber\$ visible, bootable and give and give it the Windows disk letter \$pepartletter\$ by using diskpart or powershell

- Parameter: \$disknumber\$
 - Type: String - Calltype: CallByValue
 - Parameter \$disknumber\$ Description: Number of the disk where we look for the partition
 - Parameter \$disknumber\$ Advice: First disk = 0
- Parameter: \$partitionnumber\$
 - Type: String - Calltype: CallByValue
 - Parameter \$partitionnumber\$ Description: Number of the partition on the given disk
 - Parameter \$partitionnumber\$ Advice: First partition = 1
- Parameter: \$pepartletter\$
 - Type: String - Calltype: CallByValue
 - Parameter \$pepartletter\$ Description: Windows disk letter that the given partition should have
- Parameter: \$usegpt\$
 - Type: String - Calltype: CallByValue
 - Parameter \$usegpt\$ Description: Should we expect GPT or MBR partitions (*true* or *false*)
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only, works in PE
- References: [\[getDiskUuid\]](#)

- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
Message "Enable PE partition..."
if not(stringToBool(enablePEPartition($disknumber$, $swapPartitionNumber$, $pepartletter$, $
  useGpt$)))
  LogError "Could not activate PE partition ..."
  isFatalError "failed not activate PE partition"
endif

if not (isDriveReady($pePartLetter$))
  logerror "PE drive "+$pePartLetter$+": not ready"
  isFatalError "PE drive "+$pePartLetter$+": not ready"
  set $errorList$ = addtoList($errorList$, " failed pe_drive_ready")
  set $fatal_error$ = "true"
endif
```

16.1.3 Documentation of opsi library: uib_exitcode.opsiscript

- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.1 Documentation of local function isMsiExitcodeFatal

Definition

```
isMsiExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ :
string) : string
```

Description

Evaluates the given \$exitcode\$ as MSI Error and gives back a resulting error message on \$ErrorString\$. If the Error require a reboot the given parameter \$allowRebootRequest\$ is used to decide if a *ExitWindows /Reboot* is called or not

- Parameter: \$exitcode\$
 - Type: String - Calltype: CallByValue
 - Parameter \$exitcode\$ Description: Exit code given by msifixec
- Parameter: \$allowrebootrequest\$
 - Type: String - Calltype: CallByValue
 - Parameter \$allowrebootrequest\$ Description: Should we call *ExitWindows /Reboot* if the exit code require this (*true* or *false*)
- Parameter: \$errorstring\$
 - Type: String - Calltype: CallByReference

- Parameter `$errorstring$` Description:
Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"
- SpecialCase: Winows only
- References: [\[isMsExitcodeFatal_short\]](#) [\[isAdvancedMsiExitcodeFatal\]](#) [\[isInnoExitcodeFatal\]](#) [\[isInstallshieldExitcodeFatal\]](#) [\[isNsisExitcodeFatal\]](#)
- Links: [http://msdn.microsoft.com/en-us/library/aa372835\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx) <http://msdn.microsoft.com/en-us/library/aa368542.aspx>
- Author: detlef oertel
- Date: 19.9.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

Example:

```
[actions]
DefVar $ExitCode$
DefVar $ErrorString$
(...)
set $ExitCode$ = getlastexitcode
if stringtobool(isMsiExitcodeFatal($exitcode$, "true", $ErrorString$ ))
    LogError $ErrorString$
    isfatalerror $ErrorString$
else
    Comment $ErrorString$
endif
```

16.1.3.2 Documentation of local function `isMsExitcodeFatal_short`

Definition

```
isMsExitcodeFatal_short($exitcode$ : string, $allowRebootRequest$ : string, ref
$ErrorString$ : string) : string
```

Description

Evaluates the given `$exitcode$` as MS Error and gives back a resulting error message on `$ErrorString$` if the exit code is well known. For full list of exit codes use `isMsiExitcodeFatal`. If the Error requires a reboot the given parameter `$allowRebootRequest$` is used to decide if a *ExitWindows /Reboot* is called or not.

- Parameter: `$exitcode$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$exitcode$` Description:
Exit code given by ms
- Parameter: `$allowrebootrequest$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$allowrebootrequest$` Description:
Should we call *ExitWindows /Reboot* if the exit code requires this (*true* or *false*)
- Parameter: `$errorstring$`
 - Type: `String` - Calltype: `CallByReference`

- Parameter `$errorstring$` Description:
Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"
- SpecialCase: Winows only
- References: [\[isMsiExitcodeFatal\]](#) [\[isAdvancedMsiExitcodeFatal\]](#) [\[isInnoExitcodeFatal\]](#) [\[isInstallshieldExitcodeFatal\]](#) [\[isNsisExitcodeFatal\]](#)
- Links: [http://msdn.microsoft.com/en-us/library/aa372835\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx) <http://msdn.microsoft.com/en-us/library/aa368542.aspx>
- Author: detlef oertel
- Date: 19.9.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.3 Documentation of local function `isAdvancedMsiExitcodeFatal`

Definition

```
isAdvancedMsiExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref
$errorString$ : string) : string
```

Description

Please note: Import complete file `uib_exitcode` (not only `isAdvancedMsiExitcodeFatal`) Evaluates the given `$exitcode$` as `AdvancedMsi` Error and gives back a resulting error message on `$ErrorString$` It is also checked if the exit code is one from the embedded msi. There for is `isMsiExitcodeFatal` used If the Error require a reboot the given parameter `$allowRebootRequest$` is used to decide if a *ExitWindows /Reboot* is called or not

- Parameter: `$exitcode$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$exitcode$` Description:
Exit code given by `AdvancedMsi`
- Parameter: `$allowrebootrequest$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$allowrebootrequest$` Description:
Should we call *ExitWindows /Reboot* if the exit code require this (*true* or *false*)
- Parameter: `$errorstring$`
 - Type: `String` - Calltype: `CallByReference`
 - Parameter `$errorstring$` Description:
Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"
- SpecialCase: Winows only
- References: [\[isMsiExitcodeFatal\]](#) [\[isAdvancedMsiExitcodeFatal\]](#) [\[isInnoExitcodeFatal\]](#) [\[isInstallshieldExitcodeFatal\]](#) [\[isNsisExitcodeFatal\]](#)
- Author: detlef oertel
- Date: 19.9.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.4 Documentation of local function isInnoExitcodeFatal

Definition

```
isInnoExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ :
string) : string
```

Description

Evaluates the given `$exitcode$` as Inno Error and gives back a resulting error message on `$ErrorString$`. If the Error require a reboot the given parameter `$allowRebootRequest$` is used to decide if a `ExitWindows /Reboot` is called or not

- Parameter: `$exitcode$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$exitcode$` Description: Exit code given by Inno
- Parameter: `$allowrebootrequest$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$allowrebootrequest$` Description: Should we call `ExitWindows /Reboot` if the exit code require this (*true* or *false*)
- Parameter: `$errorstring$`
 - Type: `String` - Calltype: `CallByReference`
 - Parameter `$errorstring$` Description: Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"
- SpecialCase: Winows only
- References: [\[isMsiExitcodeFatal\]](#) [\[isAdvancedMsiExitcodeFatal\]](#) [\[isInnoExitcodeFatal\]](#) [\[isInstallshieldExitcodeFatal\]](#) [\[isNsisExitcodeFatal\]](#)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.5 Documentation of local function isInstallshieldExitcodeFatal

Definition

```
isInstallshieldExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref
$ErrorString$ : string) : string
```

Description

Please note: Import complete file `uib_exitcode` (not only `isAdvancedMsiExitcodeFatal`) Evaluates the given `$exitcode$` as Installshield Error and gives back a resulting error message on `$ErrorString$`. It is also checked if the exit code is one from the embedded msi. There for `isMsiExitcodeFatal` is used. If the Error require a reboot the given parameter `$allowRebootRequest$` is used to decide if a `ExitWindows /Reboot` is called or not

- Parameter: `$exitcode$`
 - Type: `String` - Calltype: `CallByValue`

- Parameter `$exitcode$` Description:
Exit code given by Installshield
- Parameter: `$allowrebootrequest$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$allowrebootrequest$` Description:
Should we call *ExitWindows /Reboot* if the exit code require this (*true* or *false*)
- Parameter: `$errorstring$`
 - Type: `String` - Calltype: `CallByReference`
 - Parameter `$errorstring$` Description:
Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"
- SpecialCase: Winows only
- References: [\[isMsiExitcodeFatal\]](#) [\[isAdvancedMsiExitcodeFatal\]](#) [\[isInnoExitcodeFatal\]](#) [\[isInstallshieldExitcodeFatal\]](#) [\[isNsisExitcodeFatal\]](#)
- Author: detlef oertel
- Date: 19.9.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.6 Documentation of local function `isNsisExitcodeFatal`

Definition

```
isNsisExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string
```

Description

Evaluates the given `$exitcode$` as Nsis Error and and gives back a resulting error message on `$ErrorString$` If the Error require a reboot the given parameter `$allowRebootRequest$` is used to decide if a *ExitWindows /Reboot* is called or not

- Parameter: `$exitcode$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$exitcode$` Description:
Exit code given by Nsis
- Parameter: `$allowrebootrequest$`
 - Type: `String` - Calltype: `CallByValue`
 - Parameter `$allowrebootrequest$` Description:
Should we call *ExitWindows /Reboot* if the exit code require this (*true* or *false*)
- Parameter: `$errorstring$`
 - Type: `String` - Calltype: `CallByReference`
 - Parameter `$errorstring$` Description:
Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"

- SpecialCase: Winows only
- References: [\[isMsiExitcodeFatal\]](#) [\[isAdvancedMsiExitcodeFatal\]](#) [\[isInnoExitcodeFatal\]](#) [\[isInstallshieldExitcodeFatal\]](#) [\[isNsisExitcodeFatal\]](#)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.4 Documentation of opsi library: uib_strlistutils.opsiscript

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

16.1.4.1 Documentation of local function compareLists

Definition

```
compareLists($list1$ : stringlist, $list2$ : stringlist) : string
```

Description

Checks if to string lists are completely identic or not. The check is not case sensitive.

- Parameter: `$list1$`
 - Type: `Stringlist` - Calltype: `CallByValue`
 - Parameter `$list1$` Description: First stringlist to compare
- Parameter: `$list2$`
 - Type: `Stringlist` - Calltype: `CallByValue`
 - Parameter `$list2$` Description: Second stringlist (to compare with first)
- Returns: Returns string "true" if the given lists are identic. Returns string "false" if the given lists are not identic
- OnError: Returns string "false"
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
set $tmpList$ = getListContainingList($baseproducts$, $resultlist$)
if stringToBool(compareLists($tmpList$, $baseproducts$))
    comment "check installed products successful finished"
else
    LogError "check installed products failed"
    set $errorList$ = addtoList($errorList$, "check installed products failed")
    ;set $fatal_error$ = "true"
endif
```