Table of Contents

1. Copyright .................................................................................................................. 2
2. opsi-script reference card (4.12.4.35) ................................................................. 3
   2.1. Global text constants ....................................................................................... 3
       2.1.1. System directories .............................................................................. 3
       2.1.2. Common (AllUsers) directories [W]: ........................................... 3
       2.1.3. Default User directories [W]: ......................................................... 4
       2.1.4. Current user directories [W]: .......................................................... 4
       2.1.5. /AllUserProfiles directory constants [W/L/M]: ..................... 4
       2.1.6. opsi-script Path and Directories [W/L/M]: ................................... 4
       2.1.7. Network informations [W/L/M]: ....................................................... 5
       2.1.8. Service Data [W/L/M] .................................................................... 5
       2.1.9. Functions to handle constants [W/L/M] ............................................ 5
2.2. In Primary Sections .............................................................................................. 6
       2.2.1. Kinds of Primary Sections [W/L/M]: .................................................. 6
       2.2.2. opsi-script control [W/L/M]: ............................................................ 6
       2.2.3. Variables [W/L/M]: ........................................................................ 7
       2.2.4. Functions ......................................................................................... 7
2.3. Secondary Sections ............................................................................................. 16
       2.3.1. Winbatch [W/L/M] ........................................................................ 16
       2.3.2. DosBatch and DosInAnlcon (ShellBatch and ShellInAnlcon) [W/L/M] 17
       2.3.3. ExecWith [W/L/M] ......................................................................... 18
       2.3.4. Files [W/L/M] ................................................................................. 18
       2.3.5. Registry [W] ..................................................................................... 20
       2.3.6. Patches [W/L/M] ............................................................................ 21
       2.3.7. PatchTextFile [W/L/M] ................................................................. 21
       2.3.8. LinkFolder [W/L/M] .................................................................... 22
       2.3.9. OpiserviceCall [W/L/M] ................................................................. 23
       2.3.10. PatchHosts [W/L/M] ................................................................. 23
       2.3.11. XML2 Sections [W/L/M] ............................................................... 24
       2.3.12. XMLPatch [W] .......................................................................... 25
       2.3.13. ExecPython [W/L/M] ................................................................. 25
       2.3.14. LdapSearch [WLM] .................................................................... 25
2.4. By Topic ............................................................................................................... 25
       2.4.1. Compare related functions [W/L/M] .................................................. 25
       2.4.2. Crypt / Hash related functions [W/L/M] ......................................... 26
       2.4.3. Defined Functions and Libraries [W/L/M] ....................................... 26
       2.4.4. Encoding related functions [W/L/M] ............................................. 27
       2.4.5. Error / Warning related functions [W/L/M] .................................... 27
       2.4.6. File related functions [W/L/M] .......................................................... 28
       2.4.7. Ini file related functions [W/L/M] ..................................................... 29
       2.4.8. Interaction [W/L/M] ..................................................................... 30
       2.4.9. License Management related functions [W/L/M] .......................... 30
       2.4.10. Linux specific functions [W/L/M] .................................................. 30
       2.4.11. Logging related functions [W/L/M] ................................................ 31
       2.4.12. macOS specific functions [W/L/M] .............................................. 32
       2.4.13. Network related functions [W/L/M] ............................................. 33
       2.4.14. Number related functions [W/L/M] ............................................. 33
       2.4.15. Operating System related functions [W/L/M] ............................. 33
       2.4.16. opiservicecall and json Related functions [W/L/M] .................. 34
       2.4.17. opsi related functions [W/L/M] ..................................................... 35
9. Syntax and Meaning of Primary Sections of a opsi-script Script [W/L/M]

9.1. Primary Sections [W/L/M] .............................................................. 71

9.2. Parametrizing opsi-script [W/L/M] .............................................. 71
  9.2.1. Specification of Logging Level [W/L/M] ................................. 71
  9.2.2. Required opsi-script Version [W/L/M] ............................... 72
  9.2.3. Reacting on Errors [W/L/M] .................................................. 72
  9.2.4. Staying On Top [W] .............................................................. 73
  9.2.5. Show window mode / Skin / Activity [W/L/M] .................... 74

9.3. String Expressions, String Values, and String Functions [W/L/M]
  9.3.1. Elementary String Values .................................................. 75
  9.3.2. Strings in Strings (Nested String Values) ............................ 75
  9.3.3. String Concatenation ......................................................... 75
  9.3.4. String Variables ............................................................... 75
  9.3.5. String Functions which Return the OS Type ......................... 76
  9.3.6. String Functions for Retrieving Environment or Command Line Data [W/L/M] .......................... 77
  9.3.7. Reading Values from the Windows Registry and Transforming Values into Registry Format [W] ............................. 78
  9.3.8. Reading Values from ini files [W/L/M] ................................ 80
  9.3.9. Reading Product Properties [W/L/M] .................................. 80
  9.3.10. Retrieving Data from etc/hosts [W/L/M] ............................ 81
  9.3.11. String Handling [W/L/M] .................................................. 81
  9.3.12. Other String Functions .................................................... 88
  9.3.13. (String-) Functions for Licence Management [W/L/M] ........ 95

9.4. String List Functions and String List Processing [W/L/M]
  9.4.1. Info Maps ........................................................................ 97
  9.4.2. Producing String Lists from Strings [W/L/M] ...................... 114
  9.4.3. Loading Lines of a Text File into a String List .................... 115
  9.4.4. Simple String Values generated from String Lists or Files [W/L/M] ......................................................... 115
  9.4.5. Producing String Lists from opsi-script Sections [W/L/M] .... 117
  9.4.6. Stringlists from the registry [W] .......................................... 118
  9.4.7. Stringlists from the Product Properties [W/L/M] ................ 120
  9.4.8. Other String Lists [W/L/M] ............................................... 121
  9.4.9. Transforming String Lists [W/L/M] ..................................... 122
  9.4.10. Iterating through String Lists [W/L/M] ............................ 127
  9.5. opsiservicecall and json Related functions [W/L/M] .............. 127
  9.6. Calculating with numbers [W/L/M] ......................................... 130
  9.7. XML2 Functions [W/L/M] ....................................................... 133
  9.8. Regular expression related functions [W/L/M] ....................... 134
  9.9. URL related functions [W/L/M] .............................................. 141
  9.10. Network related functions [W/L/M] ...................................... 142


9.13. Control the logging .................................................................. 149


9.15. Commands for userLoginScripts / User Profile Management 152

9.16. for to do loop .......................................................................... 153

9.17. Switch / Case Statement [W/L/M] ............................................ 154

9.18. Conditional Statements (if Statements) ................................. 155
  9.18.1. General Syntax ................................................................. 156
  9.18.2. Boolean Expressions ........................................................ 156

9.19. Include Commands ................................................................. 163
  9.19.1. Include Commands: Syntax ............................................. 163
  9.19.2. Include Commands: Library ................................ .......... 165
9.20. Subprogram Calls ................................................. 166
  9.20.1. Syntax of Procedure Calling ................................ 167
9.21. Controlling Reboot .............................................. 168
9.22. Abort script and keep track of failed installations .......... 169
9.23. Local functions [W/L/M] ...................................... 171
  9.23.1. Concept ................................................ 171
  9.23.2. Syntax ............................................... 172
  9.23.3. Examples ........................................... 172
9.24. Import of libraries of functions [W/L/M] ..................... 175
10. Secondary Sections ............................................. 176
10.1. Calling secondary sections .................................... 176
10.2. Files Sections .................................................. 177
  10.2.1. Example .............................................. 177
  10.2.2. Modifier ............................................. 177
  10.2.3. Commands ............................................ 178
10.3. Patches-Sections [W/L/M] .................................... 181
  10.3.1. Example .............................................. 181
  10.3.2. Commands ............................................ 182
10.4. PatchHosts Sections [W/L/M] ................................ 183
10.5. IdapiConfig Sections ....................................... 184
10.6. PatchTextFile Sections [W/L/M] ............................. 184
  10.6.1. Parameter ............................................ 184
  10.6.2. Commands ............................................ 184
  10.6.3. Examples ............................................ 186
10.7. LinkFolder Sections [W/L/M] ................................ 186
  10.7.1. LinkFolder Sections in Windows ......................... 186
  10.7.2. Examples ............................................ 188
  10.7.3. LinkFolder-Sections in Linux .......................... 188
10.8. XML2 Section [W/L/M] ...................................... 189
  10.8.1. XML structure and wording ............................. 189
  10.8.2. CallParameter ...................................... 190
  10.8.3. Commands ............................................ 190
  10.8.4. XML2 Examples ..................................... 192
10.9. XMLPatch Sections [W] ...................................... 197
  10.9.1. Parameter ............................................ 197
  10.9.2. Structure of a XML Document ......................... 197
  10.9.3. Options for Selection a Set of Elements ................ 198
  10.9.4. Patch Actions ...................................... 199
  10.9.5. Returning Lists to the Caller .......................... 200
  10.9.6. Examples ............................................ 201
10.10. ProgmanGroups Sections .................................... 201
10.11. WinBatch-Sections [W/L/M] .................................. 201
  10.11.1. Call Parameter (Modifier) ............................. 201
  10.11.2. Examples ............................................ 204
10.12. DOSBatch/DosInAnIcon (ShellBatch/ShellInAnIcon) Sections [W/L/M] 204
  10.12.1. Parameter ............................................ 205
  10.12.2. Catch the output ................................... 207
  10.12.3. Examples ............................................ 208
10.13. Registry-Sections [W] ...................................... 208
  10.13.1. Examples ............................................ 208
  10.13.2. Call Parameters ..................................... 208
  10.13.3. Commands ............................................ 208
  10.13.4. Registry Sections to Patch All NTUser.dat .......... 211
16.1.3. Documentation of opsi library: uib_exitcode.opsiscript .................................................. 257
16.1.4. Documentation of opsi library: uib_strlistutils.opsiscript .................................................. 264
16.1.5. Documentation of opsi library: uib_driver_install.opsiscript ............................................... 264
16.1.6. Documentation of opsi library: uib_reboot_lib.opsiscript ..................................................... 266
16.1.7. Documentation of opsi library: uib_state_and_condition.opsiscript ....................................... 266
16.1.8. Documentation of opsi library: uib_lin_install.opsiscript .................................................... 269
16.1.9. Documentation of opsi library: uib_macosinstalllib.opsiscript ............................................... 277
For Windows [W], Linux [L] and macOS [M]
Chapter 1. Copyright

The Copyright of this manual is held by uib gmbh in Mainz, Germany.

This manual is published under the creative commons license Attribution - ShareAlike (by-sa).

A description of the license can be found here:
https://creativecommons.org/licenses/by-sa/3.0/

The legally binding text of the license can be found here:
https://creativecommons.org/licenses/by-sa/3.0/legalcode

Most parts of the opsi software is open source.
Not open source are the parts of the source code which contain new extensions, that are still under cofunding, which have not been paid off yet. See also: opsi cofunding projects

All of the open source code is published under the AGPLv3.

The legally binding text of the AGPLv3 license can be found here: http://www.gnu.org/licenses/agpl-3.0-standalone.html

Information about the AGPL: http://www.gnu.org/licenses/agpl-3.0.en.html

For licenses to use opsi in the context of closed source software, please contact uib gmbh.

The names opsi, opsi.org, open pc server integration and the opsi logo are registered trademarks of uib gmbh.
Chapter 2. opsi-script reference card (4.12.4.35)

For Windows [W], Linux [L] and MacOS [M]

2.1. Global text constants

2.1.1. System directories

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: Section 8.2.3.1, “Base system directories [W]”

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 Setting\All Users\Startmenu
NT6: C:\ProgramData\Microsoft\Windows\Start Menu

%CommonAppdataDir%:
NT5: c:\Documents and Setting\All Users\Application Data
NT6: C:\ProgramData

%CommonDesktopDir%:
NT5: c:\Documents and Setting\All Users\Desktop
NT6: C:\Users\Public\Desktop

%CommonStartupDir%:
NT5: c:\Documents and Setting\All Users\Autostart
NT6: C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp

%CommonProgramsDir%

see also: Section 8.2.3.2, “Common (AllUsers) directories [W]”
2.1.3. Default User directories [W]:

%DefaultUserProfileDir% // since 4.11.1.1

see also: Section 8.2.3.3, “Default User Directory [W]”

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: Section 8.2.3.4, “Current (logged in or usercontext) user directories [W]”

2.1.5. /AllUserProfiles directory constants [W/L/M]:

%UserProfileDir%
or%CurrentProfileDir% // since 4.11.2.1
NT5: c:\Documents and Settings\%USERNAME%
NT6: c:\users\%USERNAME%

see also: Section 8.2.3.5, “/AllUserProfiles (/AllNtUserProfiles) directory constants [W/L/M]”

2.1.6. opsi-script Path and Directories [W/L/M]:

%ScriptPath% or %ScriptDir%

%RealScriptPath% (since 4.12.4.21)

%ScriptDrive%

%OpsiscriptDir% (since 4.12.3.6), %WinstDir%

%OpsiscriptVersion% (since 4.12.3.6), %WinstVersion% (since 4.10.8.3)

%OpsiscriptProcname% (since 4.12.4.35)

%Logfile%

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

%opsiTmpDir%: c:\opsi.org\tmp // since 4.11.4.3
2.1.7. Network informations [W/L/M]:

%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 : Section 8.2.5, “Network Information [W/L/M]”

2.1.8. Service Data [W/L/M]

%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 : Section 8.2.6, “Data for and from opsi service [W/L/M]”

2.1.9. Functions to handle constants [W/L/M]

replaceOpsiConstants(<string list>) : stringlist //since 4.12.3.6 [W/L/M] see also : [replaceOpsiConstants_list]

replaceOpsiConstants(<string>) : string //since 4.12.3.6 [W/L/M] see also : [replaceOpsiConstants_string]
2.2. In Primary Sections

2.2.1. Kinds of Primary Sections [W/L/M]:

[Initial]

[Actions]

[sub<identifier>]

sub <file name>

[ProfileActions] [W]

see also: Chapter 9, Syntax and Meaning of Primary Sections of a opsi-script Script [W/L/M]

2.2.2. opsi-script control [W/L/M]:

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

LogLevel (deprecated) see also: Section 9.2.1, “Specification of Logging Level [W/L/M]”

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

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]

forceLogInAppenMode = <boolean value> // (default=false); if true log will be send in append mode . //since 4.12.3.6 see also: [forceLogInAppenMode]

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

include <scriptpath>\test-files\10lines.txt" *$"

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

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

asConfidential ( <secret stringlist expression> ) : stringlist // since 4.12.4.15 [W/L/M] see also: asConfidential_list

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

runningInWanMode : boolean // since 4.12.4.17 [W/L/M] see also: runningInWanMode

2.2.3. Variables [W/L/M]:

Strings

DefVar <variable name>
: since 4.12.4.32 also possible:
DefVar <variable name> [= <initial value>]

Set <variable name> = <value>

see also: Section 8.3, “String (or Text) Variables [W/L/M]”

Stringlists

DefstringList <variable name> ; since 4.12.4.32 also possible:
DefstringList <variable name> [= <initial value>]

see also: Section 8.4, “Stringlist Variables [W/L/M]”

2.2.4. Functions

String functions
Important

**GetOS** // Linux or Windows_NT [W/L/M] see also : [GetOS]

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

**GetMsVersionInfo** // Windows Version Information [W] see also : [GetMsVersionInfo]

**GetMsVersionName** // Windows marketing Version // since 4.12.4.35 [W] see also : [GetMsVersionName]

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

**getOSArchitecture** // OS Architecture (x86_32 / x86_64 / arm_64) // since 4.12.4.17 [W/L/M] see also : [getOSArchitecture]

**getRegistryValue** (<keystr>, <varstr>) : 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/M]

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

see also : [GetValueFromInifile]

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

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

**trim**(<string>) [W/L/M] see also : [trim]

**lower**(<string>) [W/L/M] see also : [lower]

**upper**(<string>) [W/L/M] see also : [upper]

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

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

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

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

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

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

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

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

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

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

Special: License Management

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

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

see also: [DemandLicenseKey]

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

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

see also: [FreeLicense]

Special: Usercontext / loginscripts [W]:

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]

Other

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

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

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

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

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

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

base64EncodeStr(<string>) [W/L/M] see also: [base64EncodeStr]
base64DecodeStr(<string>) [W/L/M] see also: [base64DecodeStr]

cJSONString(<string>) //since 4.10.8.3

RandomStr [W/L/M] see also: [RandomStr]

RandomStrWithParameters [W/L/M] see also: [RandomStrWithParameters]

RandomIntStr(<number str>) : string [W/L/M] see also: [RandomIntStr]

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

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

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

ParamStr [W/L/M] see also: [ParamStr]

gDiffTimeSec (Time in seconds since last marktime) //since 4.11.3 [W/L/M] see also: [gDiffTimeSec]

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/M] see also: [GetMyIpByTarget]

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

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

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

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

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

stringinput(< message str>, < boolstr confidential>) : string //since 4.12.1.2 [W/L/M] see also: [stringinput]

which(<command in path>) : string (command with path) //since 4.12.3.6 [W/L/M] see also: [which]

replaceOpsiConstants(<string>) : string //since 4.12.3.6 [W/L/M] see also: [replaceOpsiConstants_string]

Deprecated

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/M] see also: [splitString]

String list functions

Important

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

set $list1$ = splitString (*.\server\share\dir*"

see also: [splitString]
splitStringOnWhiteSpace (<string>) [W/L/M] see also: [splitStringOnWhiteSpace]

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

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

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

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

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

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

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

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

set $list$= getOutStreamFromSection ('DosInAnIcon_try')

see also: [getOutStreamFromSection]

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

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

see also: [shellCall_list]

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

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

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

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

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

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

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/M] see also: [GetProcessList]

getProductPropertyList(<propname>,<default value>) //since 4.11.3 [W/L/M] 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/M] see also: [getProfilesDirList]

listFiles (<Path>, <Searchmask>, <SearchSubDirectories>, <[Redirection]>): stringlist //since 4.12.3 [W/L/M]

see also: [listFiles]

replaceOpsiConstants(<string list>): stringlist //since 4.12.3.6 [W/L/M] see also: [replaceOpsiConstants_list]

Infomaps

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=detlefvm05
- 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(<filename>) [W] see also: [getFileInfoMap]

getProductMap // since 4.11.2.4 [W/L/M]
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/M] see also: [getHWBiosInfoMap]

editmap(<strlist>): stringlist //since 4.12.1.2 [W/L/M] see also: [editmap]

Other

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

set $list1$ = createStringList ('a','b')
Boolean operators and functions

see also: Section 9.18.2, “Boolean Expressions”

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

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

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

NOT(bool3) [W/L/M]

FileExists (<file name>) [W/L/M] see also: [FileExists]

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

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

FileExistsSysNative (<file name>) [W] see also: [FileExists]
DirectoryExists (folder path[,access str]) : boolean //since 4.12.1 [W/L/M]
sysnative is the default for <access str>. Otherwise, it can be 32bit, 64bit or sysnative see also: [DirectoryExists]

FileOrFolderExists (file or folder path[,access str]) : boolean [W/L/M] sysnative is the default for <access str>. Otherwise, it can be 32bit, 64bit or sysnative see also: [FileOrFolderExists]

cfileIsSymlink (file name) // since 4.12.4.21 [W/L/M]
see also: [fileIsSymlink]

LineExistsIn (string, file name) [W/L/M] see also: [LineExistsIn]

LineBeginning_ExistsIn (string, file name) [W/L/M] see also: [LineBeginning_ExistsIn]

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

runningAsAdmin //since 4.11.1.1 [W/L/M] see also: [runningAsAdmin]

isLoginScript //since 4.11.2.1 [W] see also: [isLoginScript]

contains (string, substr) : bool //since 4.11.3: true if substr in string [W/L/M] see also: [contains]

isNumber (str) //since 4.11.3: true if str represents an integer [W/L/M] 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/M] see also: [runningInPE]

runningInWAnMode //since 4.12.4.16: true: if opsiserver = localhost [W/L/M] see also: [runningInWAnMode]

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

runningWithGui : bool //since 4.12.3.6: true: if the running OS has a GUI (at Win+Mac always true)[M/L/W] see also: [runningWithGui]

saveTextFile (list, filename) //since 4.11.4.4: true: if list is succesfully written to file [W/L/M] see also: [saveTextFile]

saveTextFileWithEncoding (list, filename, encoding) : bool //since 4.11.6.4: true: if list is succesfully written to file [W/L/M] see also: [saveTextFileWithEncoding]

saveUnicodeTextFile (list, filename, encoding) : bool //since 4.12.4.13: true: if list is succesfully written to unicode file [W/L/M] see also: [saveUnicodeTextFile]

CompareDotSeparatedNumbers (str1, relation str, str2) //since 4.11.5.2: [W/L/M] see also: [CompareDotSeparatedNumbers_bool]
CompareDotSeparatedStrings(<str1>,<relation str>,<str2>) //since 4.11.5.2 [W/L/M] see also : [CompareDotSeparatedStrings_bool]

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

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

isPingReachable(<host>) : boolean //since 4.12.3.6 [W/L/M] see also : [isPingReachable]

isValidFQDN(<domain name>) : boolean //since 4.12.4.4 [W/L/M] see also : [isValidFQDN]

fileHasBom (<file name>) : boolean //since 4.12.4.17 [W/L/M] see also : [fileHasBom]

Misc functions

Killtask <process name> [W/L/M] see also : [KillTask]

requiredOpsiscriptVersion <relation operator> <version> //since 4.12.3.6 [W/L/M]

requiredOpsiscriptVersion >= "4.12.3.6"

see also : [requiredOpsiscriptVersion]

requiredWinstVersion <relation operator> <version> [W/L/M]

outdatet - use requiredOpsiscriptVersion see also : [requiredOpsiscriptVersion]

UpdateEnvironment //since 4.11.5 [W]:
Subsequent calls of winbatch with the parameter /RunElevated will see the changed Environment (NT6 only). see also : [UpdateEnvironment]

Flow control

if - else - endif [W/L/M] 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/M] see also : [ForToDo]

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

Example:
for %s% = "1" to "5" do sub_iteration_text

**Switch / Case Statement //since 4.11.5 [W/L/M] see also : [SwitchCase]**

Syntax:

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

Example:

```plaintext
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/M] see also : [isFatalError]**

**isFatalError <string> //since 4.11.3.2 [W/L/M] see also : [isFatalError]**

**isSuccess //since 4.11.3.7 [W/L/M] see also : [isSuccess]**

**isSuspended //since 4.11.4.1 [W/L/M] see also : [isSuspended]**

**noUpdateScript //since 4.11.3.7 [W/L/M] see also : [noUpdateScript]**

**ExitWindows /Reboot [W/L/M] see also : [Reboot]**

**ExitWindows /ImmediateReboot [W/L/M] 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/M] see also : [sleepSeconds]**

**ChangeDirectory <directory> //since 4.11.2.6 [W/L/M] see also : [ChangeDirectory]**

# 2.3. Secondary Sections

## 2.3.1. Winbatch [W/L/M]

see also : Section 10.11, “WinBatch-Sections [W/L/M]”

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/M]

see also: Section 10.12, “DOSBatch/DosInAnIcon (ShellBatch/ShelfInAnIcon) Sections [W/L/M]”

Function: Execute section via cmd.exe [W] or bash [L/M]

[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/M] // since 4.11.4.7

/WaitForProcessEnding "<program.exe>" // since 4.12.4 [W/L/M]

/TimeoutSeconds <seconds> // since 4.12.4 [W/L/M]

/RunElevated // since 4.12.4: only at >= NT6; no network access [W]

The modifiers has to be seperated by winst from the parameters.
Commands: see manual

2.3.3. ExecWith [W/L/M]

see also: Section 10.16, “ExecWith Sections [W/L/M]”

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 \textit{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" \textit{winst} /64Bit
```

Commands: see manual

2.3.4. Files [W/L/M]

see also: Section 10.2, “Files Sections”

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

options:

```
-s recursive [W/L/M]
```
-e empty Subdirectories

-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/M]

options:

-s recursive

-f force

r -del on reboot

c continue with out reboot

d [n] date

Example (do not forget the trailing Backslash):
delete -sf c:\delete_this_dir\ 

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

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 <mode> <path> //since 4.11.4.1 [L]

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

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

At Windows symlink is only available at NT6 and up.

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

move <old filename> <new filename> // since 4.11.5 [W/L/M]
zipfile <source dir> <zip file> // since 4.12.1 [W/L/M]

unzipped <zip file> <target dir> // since 4.12.1 [W/L/M]

### 2.3.5. Registry [W]

see also: Section 10.13, “Registry-Sections [W]”

Function: edit Registry

Standard method call:
```c
[Registry<identifier>]
```
works with the specified section.

Alternative method call:
```c
Registry loadUnicodeTextFile(<.reg file>) /regedit
```
import the specified <.reg file>.

Alternative method call (deprecated):
```c
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>**

```c
openkey [HKEY\Software\opsi.org]
```

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

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

Examples for registry types:

```c
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>**

```c
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/M]

see also: Section 10.3, “Patches-Sections [W/L/M]”

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/M]

see also: Section 10.6, “PatchTextFile Sections [W/L/M]”

Function: edit text files

[PatchTextFile<identifier>] <file name>

Modifier:

/AllNTUserProfiles //since 4.11.3.4 [W]

/encoding <encoding> //since 4.12.4.17 [W/L/M]

Commands:

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

preference type takes any value.
Some examples for preference types: pref, user_pref, lock_pref or lockPref.

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_StartWith <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/M]

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

2.3.8. LinkFolder [W/L/M]

see also : Section 10.7, “LinkFolder Sections [W/L/M]”

Function: Startmenu + 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] [W]
  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

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

```plaintext
[LinkFolder_configed_lin]
set_basefolder common_programs
set_subfolder ""
set_link
name: opsi-configed-Local
target: java
parameters: $parameter$
icom_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 sensitive) divided by ', '-' or 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 their 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/M]

see also : Section 10.14, “OpsiServiceCall Sections [W/L/M]”

Function: opsi-Service access

```
[OpsiServiceCall<identifier>]
```

Commands: see manual

2.3.10. PatchHosts [W/L/M]

see also : Section 10.4, “PatchHosts Sections [W/L/M]”

Function: hosts-files bearbeiten

```
[PatchHosts<identifier>]
```

Commands:

```
setaddr <hostname> <IPaddress>
```
setname <Ipaddress> <hostname>

setalias <hostname> <alias>

setalias <Ipadress> <alias>

delalias <hostname> <alias>

delalias <Ipadress> <alias>

delhost <hostname>

delhost <ipadresse>

setComment <ident> <comment>

### 2.3.11. XML2 Sections [W/L/M]

see also: Section 10.8, “XML2 Section [W/L/M]”

Function: edit XML files
since 4.12.1.0

[XML2<identifier> ]

Commands:

- **strictMode** = (true/false); Default: false

- **openNode** <xml2 path>
  
  Open the given Path as actual node. If the path does not exist it will be created

- **SetAttribute** <attr name> <attr value>
  
  At the actual node set <attr value> as value of <attr name>. If <attr name> not exists, it will be created.

- **AddAttribute** <attr name> <attr value>
  
  If at the actual node the attribute <attr name> not exists, it will be created with <attr value> as value. If <attr name> still exists, nothing will be changed.

- **DeleteAttribute** <attr name>
  
  If at the actual node the attribute <attr name> exists, it will be deleted.

- **addNewNode** <node name>
  
  Create at the actual node a new sub node <node name> and make this new node to the actual node.

- **setText** <string>

- **DeleteNode** <xml2 path>

- **gotoParentNode**
  
  Make the parent node to the actual node.

- **rootNodeOnCreate** = <node name> // since 4.12.4.27
  
  If the file not exists it will be created with <node name> as root node name.

- **setNodePair** <keyNodeName> <keyNodeTextContent> <valueNodeName> <valueNodeTextContent> // since 4.12.4.28
  
  Creates a <dict> entry like it is used in the Apple info.plist files:

Some notes to the command parameters:

- `<xml2 path> strictMode =false`
  
  A line of xml node names with optional one attribute separated by `//` + Example: `node_level-1_number-1 // node_level-2_B color="green"`
see also: Section 2.4.25, “XML related functions (XML2) [W/L/M]:”

see also: Section 9.7, “XML2 Functions [W/L/M]”

2.3.12. XMLPatch [W]

see also: Section 10.9, “XMLPatch Sections [W]”

Function: edit XML files
Depricated: please use xml2 sections: Section 2.3.11, “XML2 Sections [W/L/M]”
and xml2 functions: Section 2.4.25, “XML related functions (XML2) [W/L/M]:”

[XMLPatch<identifier>]

Commands: see manual

2.3.13. ExecPython [W/L/M]

see also: Section 10.15, “ExecPython Sections [W/L/M]”

Function: Execute section via python interpreter

[ExecPython<identifier>]

Commands: see manual

2.3.14. LdapSearch [WLM]

see also: Section 10.17, “LDAPsearch Sections [WLM]”

Function: read from LDAP

[LdapSearch<identifier>]

Commands: see manual

2.4. By Topic

2.4.1. Compare related functions [W/L/M]

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

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

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

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

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

 stringToBool(<string expression: true/false>) : boolean // since 4.12.0.0 [W/L/M] see also : [stringToBool]
2.4.2. Crypt / Hash related functions [W/L/M]

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

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

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

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

RandomStr : string [W/L/M] see also : [RandomStr]

RandomIntStr(<number str>) : string [W/L/M] see also : [RandomIntStr]

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

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

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

2.4.3. Defined Functions and Libraries [W/L/M]

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 choosen 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 wether string or stringlist.

• ftype is the type of data of the function wether 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 : Section 9.23, “Local functions [W/L/M]”

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/M]

• A existing file in %ScriptPath% [W/L/M]

• A file in %opsiScriptHelperPath%\lib [W]
Is equivalent to: %ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib

* A existing file in %ScriptPath%/../lib [W/L/M]
* 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 : Section 9.24, “Import of libraries of functions [W/L/M]”

### 2.4.4. Encoding related functions [W/L/M]

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

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

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

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

```plaintext
fileHasBom (<file name>) : boolean //since 4.12.4.17 [W/L/M] see also : [fileHasBom]
```

```plaintext
loadUnicodeTextFile (<file name>) : stringlist [W/L/M] see also : [loadUnicodeTextFile]
```

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

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

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

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

see also : opsi-script encoding

### 2.4.5. Error / Warning related functions [W/L/M]

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

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

see also : [ScriptErrorMessages]

see also : [opsi-script-configs_ScriptErrorMessages]

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

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

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

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

```plaintext
isFatalError [W/L/M] see also : [isFatalError]
```

```plaintext
isFatalError <string> //since 4.11.3.2 [W/L/M] see also : [isFatalError]
```

```plaintext
markErrorNumber see also : [markErrorNumber]
```
2.4.6. File related functions [W/L/M]

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/M] see also : [ getLastExitCode ]

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

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

getServiceErrorClass : string see also : [ getServiceErrorClass ]

getServiceErrorMessage : string see also : [ getServiceErrorMessage ]

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

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

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

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

loadTextFileWithEncoding( <filename> , <encoding> ) : stringlist //since 4.11.5 [W/L/M] see also : [loadTextFileWithEncoding]

fileHasBom (<file name>) : boolean //since 4.12.4.17 [W/L/M] see also : [fileHasBom]

FileExists (<file name>) : bool [W/L/M] 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]

DirectoryExists (<folder path> [,<access str>]) : boolean //since 4.12.1 [W/L/M]
sysnative is the default for <access str>. Otherwise, it can be 32bit, 64bit or sysnative see also : [DirectoryExists]

FileOrFolderExists (<file or folder path> [,<access str>]) : boolean [W/L/M] sysnative is the default for <access str>. Otherwise, it can be 32bit, 64bit or sysnative see also : [FileOrFolderExists]

fileIsSymlink (<file name>) : bool // since 4.12.4.21 [W/L/M]
see also : [fileIsSymlink]

resolveSymlink (<file name>) : <file name> // since 4.12.4.21 [W/L/M]
see also : [resolveSymlink]

listFiles (<Path>, <Searchmask> , <SearchSubDirectories> , <[Redirection]> ) : stringlist //since 4.12.3 [W/L/M]

Set $Filelist$ = listFiles("C:\windows\system32","*.Devices.*.dll","False","64bit")

see also : [listFiles]
forcePathDelims ((<path string>) : <path string> // since 4.12.4.21 [W/L/M] see also: [forcePathDelims]

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

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

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

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

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

ExtractFileExtension(<path>) : string [W/L/M] //since 4.12.1 see also: [ExtractFileExtension]

ExtractFileName(<path>) : string [W/L/M] //since 4.12.1 see also: [ExtractFileName]

see also: Section 2.3.4, “Files [W/L/M]”

see also: Section 2.3.7, “PatchTextFile [W/L/M]”

2.4.7. Ini file related functions [W/L/M]

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

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

see also: [GetValueFromInifile]

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

GetSectionFromInifile(<ini-file>) : stringlist [W/L/M] see also: [GetSectionFromInifile]

getValue(key string, hash string list) : string [W/L/M] see also: [getValue]

getValueBySeparator(key string, separator string, hash string list) : string //since 4.11.2.1 [W/L/M] see also: [getValueBySeparator]

getValueFromFile(key string, file name) : string //since 4.11.4.4 [W/L/M] see also: [getValueFromFile]

getValueFromFileBySeparator(key string, separator string, file name) : string //since 4.11.4.4 [W/L/M] see also: [getValueFromFileBySeparator]

see also: Section 2.3.6, “Patches [W/L/M]”
2.4.8. Interaction [W/L/M]

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/M] 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]

stringinput(< message str>,< boolstr confidential>) : string //since 4.12.1.2 [W/L/M] see also : [stringinput]

editmap(< strlist>) : stringlist //since 4.12.1.2 [W/L/M] see also : [editmap]

2.4.9. License Management related functions [W/L/M]

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. Linux specific functions [W/L/M]

GetOS : string // Linux or Windows_NT or macos [W/L/M] see also : [GetOS]

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

getLinuxVersionMap : stringlist //since 4.11.4 [L] see also : [getLinuxVersionMap]

chmod in Files sections [L/M] see also : Section 10.2.3, “Commands”

waitForPackageLock(<wait_seconds>,<abort_on_timeout>) : bool // since 4.11.6.1 [L] see also : [waitForPackageLock]

importlib "uib_lin_install":

cleanupPackageSystem : void //since 4.13.4 [L] see also : [cleanupPackageSystem]

installupdates : string //since 4.13.4 [L] see also : [installupdates]
debinstall($packagelist$ : stringlist) : string //since 4.13.4 [L] see also : [debinstall]

redinstall($packagelist$ : stringlist) : string //since 4.13.4 [L] see also : [redinstall]

suseinstall($packagelist$ : stringlist) : string //since 4.13.4 [L] see also : [suseinstall]

ucsinstall($packagelist$ : stringlist) : string //since 4.13.4 [L] see also : [ucsinstall]

genericLinInstall($packagelist$ : stringlist) : string
see also : [genericLinInstall]

linuxInstallOneOf($packagelist$ : stringlist) : string
see also : [genericLinInstall]

isOneInstalled($packagelist$ : stringlist) : string
see also : [linuxInstallOneOf]

linuxInstallOneFile($packagefile$ : string) : string
see also : [linuxInstallOneFile]

linuxRemoveOnePackage($packagename$ : string) : string
see also : [linuxRemoveOnePackage]

linuxRemoveOneOf($packagelist$ : stringlist) : string
see also : [linuxRemoveOneOf]

2.4.11. Logging related functions [W/L/M]

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

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/M] see also : [includelog]

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

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

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

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

asConfidential( <secret stringlist expression> ) : stringlist //since 4.12.4.15 [W/L/M] see also : [asConfidential_list]

forceLogInAppenMode = <boolean value> // (default=false); if true log will be send in append mode . //since 4.12.3.6 see also : [forceLogInAppenMode]
opsi-configs

see also: Section 5.2, “Central configuration via opsi Configs (Host Parameter)"

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 minimum log level; 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]

opsi-script.global.SuppressSystemEncodingWarning : boolean ; suppress some encoding warnings; default: false
see also: [opsi-script-configs_SupressSystemEncodingWarning]

opsi-script.global.ReverseProductOrderByUninstall : boolean ; product list is reordered for uninstall; default: false
see also: [opsi-script-configs_ReverseProductOrderByUninstall]

opsi-script.global.log_rotation_count : boolean ; number of opsi-script.log backups; default: 8
see also: [opsi-script-configs_log_rotation_count]

opsi-script.global.configs_writeProductLogFile : boolean ; logs for every single opsi product; default: false
see also: [opsi-script-configs_writeProductLogFile]

2.4.12. macOS specific functions [W/L/M]

GetOS : string // Linux or Windows_NT or macos [W/L/M] see also: [GetOS]

getMacosVersionInfo : string //macOS Version Information //since 4.12.1.0 [M] see also: [getMacosVersionInfo]

getMacosVersionMap : stringlist //macOS Version map //since 4.12.1.0 [M] see also: [getMacosVersionMap]

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

getOSArchitecture // OS Architecture (x86_32 / x86_64/ arm_64) //since 4.12.4.17 [W/L/M]
see also: [getOSArchitecture]

importlib "uib macOSinstalllib":

install_macos_app($myapp$ : string) : string [M]
see also: [install_macos_app]

install_macos_pkg($mypkg$ : string) : string [M]
see also: [install_macos_pkg]

install_macos_dmg($mydmg$ : string) : string [M]
see also: [install_macos_dmg]

install_macos_zip($myzip$ : string) : string [M]
install_macos_generic($myfile$ : string) : string [M] see also : [install_macos_generic]

2.4.13. Network related functions [W/L/M]

GetHostName (<hostaddress>) : string [W/L/M] see also : [GetHostName]

GetHostAddr (<hostname>) : string [W/L/M] see also : [GetHostAddr]

GetMyIpByTarget (<target_ip_addr>) : string //since 4.11.3.2 /4.11.6 [W/L/M] see also : [GetMyIpByTarget]

GetIpByName (<ip_addr / ip_name>) : string //since 4.11.3.2 [W/L/M] see also : [GetIpByName]

isValidIP4 (<ip4adr>) : boolean //since 4.12.1 see also : [isValidIP4]

isValidIP4Network (<ip4adr>, <netmask>) : boolean //since 4.12.1 see also : [isValidIP4Network]

isValidIP4Host (<ip4adr>, <netmask>) : boolean //since 4.12.1 see also : [isValidIP4Host]

getIP4NetworkByAdrAndMask (<ip4adr>, <netmask>) : string //since 4.12.1 see also : [getIP4NetworkByAdrAndMask]

getDefaultNetmaskByIP4adr (<ip4adr>) : string //since 4.12.1 see also : [getDefaultNetmaskByIP4adr]

parseUrl(<url string>) : stringlist //since 4.12.1 see also : [parseUrl]

createUrl(<urlcomponents list>) : string //since 4.12.1 see also : [createUrl]

isPingReachable(<host>) : boolean /since 4.12.3.6 see also : [isPingReachable]

isValidFQDN(<domain name>) : boolean //since 4.12.4.4 [W/L/M] see also : [isValidFQDN]

2.4.14. Number related functions [W/L/M]

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

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

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

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

RandomIntStr(<number str>) : string [W/L/M] see also : [RandomIntStr]

2.4.15. Operating System related functions [W/L/M]

GetOS : string // Linux or Windows_NT or macos [W/L/M] see also : [GetOS]

GetMsVersionInfo : string //Windows Version Information [W] see also : [GetMsVersionInfo]

GetMsVersionName //Windows marketing Version //since 4.12.4.35 [W] see also : [GetMsVersionName]
2.4.16. opsiservicecall and json Related functions [W/L/M]

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

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

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

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

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

jsonAsObjectCountElements(<jsonstr>) : intstr //since 4.11.6: [W/L/M] see also : [jsonAsObjectCountElements]
jsonAsArrayGetElementByIndex(<jsonstr>, <indexstr>) : jsonstring //since 4.11.6: [W/L/M] see also : [jsonAsArrayGetElementByIndex]

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

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

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

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

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

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

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

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

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

convert2Jsonstr(<string>) //since 4.10.8.3

see also: OpsiServiceCall Section 2.3.9, “OpsiServiceCall [W/L/M]”

see also : Section 9.5, “opsiservicecall and json Related functions [W/L/M]”

see also : Section 10.14, “OpsiServiceCall Sections [W/L/M]”

2.4.17. opsi related functions [W/L/M]

getProductMap : stringlist // since 4.11.2.4 [W/L/M]
keys are: id, name, description, advice, productversion, packageversion, priority, installationstate, lastactionrequest, lastactionresult, installeddversion, installdpackage, installedmodificationtime,actionrequest see also : [getProductMap]

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

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

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

setActionProgress <string> : noresult //since 4.11.3 [W/L/M] see also : [setActionProgress]

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

replaceOpsiConstants(<string list>) : stringlist //since 4.12.3.6 [W/L/M] see also : [replaceOpsiConstants_list]

replaceOpsiConstants(<string>) : string //since 4.12.3.6 [W/L/M] see also : [replaceOpsiConstants_string]

runningInWanMode : boolean //since 4.12.4.17 [W/L/M] see also : [runningInWanMode]
2.4.18. Process and Script Related functions

Killtask <process name> : noresult //since 4.11.2.6 [W/L/M] see also : Killtask

ChangeDirectory <directory> : noresult //since 4.11.1.2; gives list of exename;pid;dom/user //since 4.11.1.2 [W/L/M] see also : ChangeDirectory

GetProcessList : stringlist //since 4.11.1.2; gives list of exename;pid;dom/user //since 4.11.1.2 [W/L/M] see also : GetProcessList

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

isProcessChildOf(<searchprocstr>, <parentprocstr>): bool //since 4.12.4.35 [W/L/M] see also : isProcessChildOf

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

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

powershellCall(<commandstr> [,<access str>=sysnative [,<policy bool str>=true] [,<optionstr> = '']]) : stringlist (output) //since 4.12.4.28 [W] see also : powershellCall_list

getOutStreamFromSection (<dos section name>) : stringlist (output) //since 4.12.4.28 [W]/M see also : powershellCall_list

set $list$= getOutStreamFromSection ('DosInAnIcon_try')

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

powershellCall(<commandstr> [,<access str>=sysnative [,<policy bool str>=true] [,<optionstr> = '']]) : noresult //since 4.12.4.28 [W] see also : powershellCall

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

powershellCall(<commandstr> [,<access str>=sysnative [,<policy bool str>=true] [,<optionstr> = '']]) : string (exitcode) //since 4.12.4.28 [W] see also : powershellCall_str

getStreamFromSection (<dos section name>) : stringlist (output) //since 4.12.4.28 [W]/M see also : powershellCall_list

see also : getOutStreamFromSection see also : Section 10.12.2, “Catch the output”

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

getLastExitCode : string (exitcode) //since 4.11.6.1 [W/L/M] see also : getLastExitCode

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

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

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

which(<command in path>) : string (command with path) //since 4.12.3.6 [W/L/M] see also : which

executeSection(<string expr with section call>) //since 4.12.3.9 [W/L/M] see also : executeSection

see also: ExecWith sections Section 2.3.3, “ExecWith [W/L/M]”

see also: ShellBatch sections Section 2.3.2, “DosBatch and DosInAnIcon (ShellBatch and ShellInAnIcon) [W/L/M]”
2.4.19. Regular expression related functions [W/L/M]:

General Info: Section 9.8, "Regular expression related functions [W/L/M]"

\[\text{isRegexMatch(<string>, <pattern>)} : \text{boolean} /\text{since 4.12.1 see also: [isRegexMatch]} \]

\[\text{getSubListByContainingRegex(<pattern>, <target list>)} : \text{stringlist} /\text{since 4.12.1 see also: [getSubListByContainingRegex]} \]

\[\text{getRegexMatchList(<pattern>, <target list>)} : \text{stringlist} /\text{since 4.12.1 see also: [getRegexMatchList]} \]

\[\text{removeFromListByContainingRegex(<pattern>, <target list>)} : \text{stringlist} /\text{since 4.12.1 see also: [removeFromListByContainingRegex]} \]

2.4.20. Registry related functions [W]

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

\[\text{GetRegistryStringValue(["key"] var"}) : \text{string [W see also: [GetRegistryStringValue]} \]

\[\text{GetRegistryStringValue32 (["key"] var"}) : \text{string [W see also: [GetRegistryStringValue32]} \]

\[\text{GetRegistryStringValue64 (["key"] var"}) : \text{string [W see also: [GetRegistryStringValue64]} \]

\[\text{GetRegistryStringValueSysNative (["key"] var"}) : \text{string [W see also: [GetRegistryStringValueSysNative]} \]

\[\text{get RegistryKeyList32(<regkey>)} : \text{stringlist} /\text{since 4.11.3 [W see also: [getRegistryKeyList32]} \]

\[\text{get RegistryKeyList64(<regkey>)} : \text{stringlist} /\text{since 4.11.3 [W see also: [getRegistryKeyList64]} \]

\[\text{get RegistryVarListSysnative(<regkey>)} : \text{stringlist} /\text{since 4.11.3 [W see also: [getRegistryVarListSysnative]} \]

\[\text{get RegistryVarList32(<regkey>)} : \text{stringlist} /\text{since 4.11.3 [W see also: [getRegistryVarList32]} \]

\[\text{get RegistryVarList64(<regkey>)} : \text{stringlist} /\text{since 4.11.3 [W see also: [getRegistryVarList64]} \]

\[\text{get RegistryVarMap32(<regkey>)} : \text{stringlist} /\text{since 4.11.3 [W see also: [getRegistryVarMap32]} \]

\[\text{get RegistryVarMap64(<regkey>)} : \text{stringlist} /\text{since 4.11.3 [W see also: [getRegistryVarMap64]} \]

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

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

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

see also: Section 2.3.5, “Registry [W]”

see also : Section 10.13, “Registry-Sections [W]”

see also : Chapter 11, 64 Bit Support on Windows [W]

2.4.21. String handling functions [W/L/M]

see also : Section 9.3, “String Expressions, String Values, and String Functions [W/L/M]”

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

```
splitString ("\server\share\dir","\")
```

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

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

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

```
set $list10 = splitString (*\server\share\dir", ")
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

EscapeString: <sequence of characters> : string // [W/L/M] see also: [EscapeString]

stringReplaceRegex (<string>, <pattern>, <replacement string>) : string //since 4.12.1 [W/L/M] see also: [stringReplaceRegex]

stringinput (<message str>, <boolstr confidential>) : string //since 4.12.1.2 [W/L/M] see also: [stringinput]

2.4.22. Stringlist handling functions [W/L/M]

see also: Section 9.4, “String List Functions and String List Processing [W/L/M]”

ggetListContaining (<list>, <search string>) : stringlist //since 4.11.3.7 [W/L/M] see also: [getListContaining]

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

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

count (<list>) : string (number) //since 4.11.3.7 [W/L/M] see also: [count]

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

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

for %s% in $list1$ do sub_test_string

see also: Section 9.4.10, “Iterating through String Lists [W/L/M]”

createStringList (<string0>, <string1> , … ) : stringlist [W/L/M]

set $list10$ = createStringList ('a','b')

see also: [createStringList]

reverse (<list>) : stringlist [W/L/M] see also: [reverse]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

jsonStringListToJsonArray(<strlist>) : jsonstr //since 4.11.6: [W/L/M] see also: [jsonStringListToJsonArray]

jsonAsObjectGetKeyList(<jsonstr>) : stringlist //since 4.11.6: [W/L/M] see also: [jsonAsObjectGetKeyList]

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

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

see also: [splitString]

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

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

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

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

gSubListByContainingRegex(<pattern>, <target list>) : stringlist //since 4.12.1 see also: [gSubListByContainingRegex]

gSubListByContainingRegex(<pattern list>, <target list>) : stringlist //since 4.12.1 see also: [gSubListByContainingRegex]

gRegexMatchList(<pattern>, <target list>) : stringlist //since 4.12.1 see also: [gRegexMatchList]

gRegexMatchList(<pattern list>, <target list>) : stringlist //since 4.12.1 see also: [gRegexMatchList]

removeFromListByContainingRegex(<pattern>, <target list>) : stringlist //since 4.12.1 see also: [removeFromListByContainingRegex]

removeFromListByContainingRegex(<pattern list>, <target list>) : stringlist //since 4.12.1 see also: [removeFromListByContainingRegex]
stringReplaceRegexInList(<target list>, <pattern>, <replacement string>) : stringlist //since 4.12.1 see also : [stringReplaceRegexInList]

editmap(< strlist>) : stringlist //since 4.12.1.2 [W/L/M] see also : [editmap]

areListsEqual(< strlist1>, <strlist2>, <flag>) : boolean see also : [areListsEqual]

GetSectionFromInifile(<ini-file>) : stringlist [W/L/M] see also : [GetSectionFromInifile]

2.4.23. Time / Date related functions [W/L/M]

sleepSeconds <Integer> or <string> : noresult [W/L/M] breaks the program execution for <string> seconds. <string> has to represent an Integer Value see also : [sleepSeconds]

markTime : noresult [W/L/M] sets a time stamp for the current system time and logs it. see also : [markTime]

diffTimeSec : string (Time in seconds since last marktime) //since 4.11.3 [W/L/M] see also : [diffTimeSec]

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

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

2.4.24. 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]

2.4.25. XML related functions (XML2) [W/L/M]:

see also : Section 9.7, “XML2 Functions [W/L/M]”

getXml2DocumentFromfile(<path to xml file>) : xml2stringlist //since 4.12.1 see also : [getXml2DocumentFromFile]

getXml2Document(<stringlist wit xml>) : xml2stringlist //since 4.12.1 see also : [getXml2DocumentFromFile]

xml2GetFirstChildNodeByName(<xml2stringlist>, <node name str>) : xml2stringlist //since 4.12.1 see also : [xml2GetFirstChildNodeByName]

gxml2GetUniqueChildNodeByName(<xml2stringlist>, <node name str>) : xml2stringlist //since 4.12.1. see also : [xml2GetUniqueChildnodeByName]

gxml2AttributeValueByKey(<xml2stringlist>, <attr name str>) : string //since 4.12.1. see also : [xml2AttributeValueByKey]

gxml2Text(<xml2stringlist>) : string //since 4.12.1. see also : [xml2Text]
see also: Section 2.3.11, “XML2 Sections [W/L/M]”

see also: Section 10.8, “XML2 Section [W/L/M]”
Chapter 3. Introduction

The open source program *opsi-script* (former at windows *opsi-winst*) serves in the context of opsi – open pc server integration (cf. www.opsi.org) – as the central function for initiating and performing the automatic software installation. It may also be used stand alone as a tool for setup programs for any piece of software.

*opsi-script* is basically an interpreter for a specific, rather simple script language which can be used to express all relevant elements of a software installation.

A software installation that is described by a *opsi-script* script and performed by executing the script has several advantages compared with installations that are managed by a group of shell commands (e. g. copy etc.):

- *opsi-script* can log very thoroughly all operations of the installation process. The support team can check the log files, and then easily detect when errors occurred or other problematic circumstances unfold.
- Copy actions can be configured with a great variety of options if existing files should be overwritten.
- Especially, it may be configured to copy files depending on their internal version.
- There are different modes to write to the Windows registry:
  - overwrite existing values
  - write only when no value exists
  - append a value to an existing value.
- The Windows registry can be patched for all users which exist on a work station (including the default user, which is used as prototype for further users).
- There is a sophisticated syntax for an integrated patching of XML configuration files.
Chapter 4. Using *opsi-script* on Linux or macOS

4.1. Introduction

As of version 4.11.4 there is a Linux port of *opsi-script*.

As of version 4.12.1 there is a macOS port of *opsi-script*.

Conditionally to the progress on porting and the differences between Linux, Windows and macOS not all functionalities are available for all operating systems.

In the following section the availability is marked as:

- [W/L/M] may be used on Windows, Linux and macOS as well
- [W] Windows only
- [L] Linux only
- [M] macOS only

4.2. Important differences and hints

*opsi-script.exe* is at Windows a GUI application which may be started with the parameter */silent* also without a GUI.

*opsi-script* is at Linux and macOS command line version which can be started without any access to a graphical display. Nevertheless this program starts a test if a access to a graphical display is possible and (if it is) starts the GUI version *opsi-script-gui*. This feature may be suppressed by calling *opsi-script* with the parameter *-silent*.

*opsi-script-gui* is a graphical version which can not be started without access to a graphical display.

At Linux and macOS the parameter delimiter is not "/" but ".". So instead of calling *opsi-script* /help you should call *opsi-script* -help at Linux and macOS.

4.3. opsi-script path at Linux

Since opsi-client-agent 4.2 you will find all components of the opsi-script program at /opt/opsi-script/.

Before opsi-client-agent 4.2:

- executable programs:
  /usr/bin/opsi-script
  /usr/bin/opsi-script-nogui
- log files directories:
  if running with root privileges: /var/log/opsi-script
  if not running with root privileges: /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

Independent of the version:


4.4. opsi-script path at macOS

You will find all components of the opsi-script program at /Applications/opsi-script/.

- log files directories:
  - if running with root privileges: /var/log/opsi-script
  - if not running with root privileges: /tmp
- config files:
  /etc/opsi-script

4.5. Path handling in opsi-script

As of version 4.11.4 for all functions that expect a path as argument, the path string is converted to a valid path for the actual operating system. This means that all path delimiters will be set OS specific. For example a path string like /home/opsiproduct\myproduct\CLIENT_DATA will be on Linux converted to /home/opsiproduct/myproduct/CLIENT_DATA. Therefore it is not possible to handle files that have a backslash in their name.

4.6. Linux specific functions

For Linux support there are the following special functions:

- `GetOS // Linux or Windows_NT or macos [W/L/M] [GetOS]`
- `getLinuxDistroType // debian or redhat or suse [L] [getLinuxDistroType]`
- `getLinuxVersionMap [L] [getLinuxVersionMap]`
- `chmod in Files sections [L/M] [chmod]`
- `waitForPackageLock(<wait_seconds>,<abort_on_timeout>) : bool [waitForPackageLock]`

In the following chapters are some commands presented, that are useful to install software on Linux. These commands are part of the opsi-script library uib_lin_install.

For an overview we start to explain the different approaches of these commands.

- distribution independent commands
  - `cleanupPackageSystem`
  - `installupdates`
- Installation of one or more packages from an online repo for one specific distribution. (This is what you want to do if you have only one distribution in your organization)
  These command take a stringlist with the name of the packages to install as argument. If you want to install only one package you may use `createStringList(<package name>)` instead of $packagelist$.
  The given package names must match to your distribution. You have to choose the command that matches your distribution family.
  - `debinstall($packagelist$ : stringlist) : string //since 4.12.4 [L]`
  - `redinstall($packagelist$ : stringlist) : string //since 4.12.4 [L]`
  - `suseinstall($packagelist$ : stringlist) : string //since 4.12.4 [L]`
  - `ucsinstall($packagelist$ : stringlist) : string //since 4.12.4 [L]`
- Installation / uninstall of one or more packages from an online repo for one specific distribution. This command is distribution
- genericLinInstall($packagelist$ : stringlist) : string
- linuxRemoveOnePackage($packagename$ : string) : string
- linuxInstallOneFile($packagefile$ : string) : string

Installation / check / uninstall of one package from an online repo for different distributions. Therefore, this one package may have different names in different distributions. All these different names are part of the $packagelist$ argument. This command is distribution independent.

- linuxInstallOneOf($packagelist$ : stringlist) : string
- isOneInstalled($packagelist$ : stringlist) : string
- linuxRemoveOneOf($packagelist$ : stringlist) : string

The details to these library commands you will find here: Documentation of opsi library: uib_lin_install.opsiscript

References to some more Linux specific library function you will find at: Linux specific functions

### 4.7. Example scripts for Linux

#### 4.7.1. Run on Linux only

```plaintext
[Actions]
DefVar $OS$
set $OS$ = GetOS
if not ($OS$ = "Linux")
    logError "Installation aborted: wrong OS version: only Linux allowed"
    isFatalError "wrong OS"
endif
```

#### 4.7.2. Which Linux Version

```plaintext
[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", $linuxInfo$)
```

### Table 1. getLinuxVersionMap Result Examples

<table>
<thead>
<tr>
<th>Distro</th>
<th>Distributor ID</th>
<th>Release</th>
<th>Codename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu Focal</td>
<td>Ubuntu</td>
<td>20.04</td>
<td>focal</td>
<td></td>
</tr>
<tr>
<td>Debian 8</td>
<td>Debian</td>
<td>8.3</td>
<td>jessie</td>
<td>Debian GNU/Linux 8.3 (jessie)</td>
</tr>
<tr>
<td>openSUSE Leap 42.1</td>
<td>SUSE LINUX</td>
<td>42.1</td>
<td>n/a</td>
<td>openSUSE Leap 42.1 (x86_64)</td>
</tr>
<tr>
<td>SLES12SP1</td>
<td>SUSE LINUX</td>
<td>12.1</td>
<td>n/a</td>
<td>SUSE Linux Enterprise Server 12 SP1</td>
</tr>
<tr>
<td>CentOS 7.0</td>
<td>CentOS</td>
<td>7.0.1406</td>
<td>Core</td>
<td>CentOS Linux release 7.0.1406 (Core)</td>
</tr>
</tbody>
</table>
### 4.7.3. ShellInAnIcon call

```
[Actions]
ShellInAnIcon_ls

[ShellInAnIcon_ls]
set -x
ls
exit $? 
```

It’s 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.

### 4.7.4. Add a repository

#### Ubuntu / Debian

```
[Actions]
DefVar $newrepo$

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

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

[ShellInAnIcon_add_rep_deb]
set ->
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_rep_deb]
FindLine_StartingWith "$newrepo$"
DeleteTheLine
GoToBottom
InsertLine "$newrepo$"

[ShellInAnIcon_add_repokey_deb]
set ->
wget --no-check-certificate -O - $newrepo$/$Release.key | apt-key add -
apt-get update
exit $? 
```

#### SUSE

```
[Actions]
DefVar $newrepo$


ShellInAnIcon_add_opsi_repository_suse

[ShellInAnIcon_add_opsi_repository_suse]
set ->
szypper --no-gpg-checks --non-interactive --gpg-auto-import-keys refresh $newrepo$
```

---

<table>
<thead>
<tr>
<th>Distro</th>
<th>Distributor ID</th>
<th>Release</th>
<th>Codename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedHat 7.0</td>
<td>RedHatEnterpriseServer</td>
<td>7.0</td>
<td>Maipo</td>
<td>Red Hat Enterprise Linux Server release 7.0 (Maipo)</td>
</tr>
<tr>
<td>UCS 4.1</td>
<td>Univention</td>
<td>4.1-1 errata122</td>
<td>Vahr</td>
<td>Univention Corporate Server 4.1-1 errata122 (Vahr)</td>
</tr>
</tbody>
</table>
CentOS / Redhat

```plaintext
[Actions]
DefVar $newrepo$

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

set ->
yum -y install wget
cd /etc/yum.repos.d
wget --no-check-certificate $newrepo$
exit $? 

PATCHTEXTFILE_ADD_REPO_REKHAT
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 "enabled=1"

ShellInAnIcon_refresh_repo_redhat
set ->
yum makecache
yum -y repolist
exit $? 
```

4.7.5. Delete a repository

Ubuntu / Debian

```plaintext
[Actions]
DeFVar $delrepos$
DefStringlist = $resultlist$
set $delrepos$ = "deb http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40/Debian_7.0/ ./"
if LineBeginning_ExistsIn($delrepos$, "/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 "$delrepos$"
DeleteThatLine
```

SUSE

```plaintext
[Actions]
DeFVar $delrepos$

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

ShellInAnIcon_del_opsi_repository_suse
set ->
syzyx --non-interactive rr $delrepos$
exit $? 
```
4.7.6. Installing a package

Generic for all supported distributions

A simple example:

A more sophisticated example:
4.8. macOS specific functions

For macOS support there are the following special functions:

- `GetOS` // `Linux` or `Windows_NT` or `macos` [W/L/M] [GetOS]
- `getMacosVersionInfo` [M] [getMacosVersionInfo]
- `getMacosVersionMap` [M] [getMacosVersionMap]
- `getOSArchitecture` // `x86_32` or `x86_64` or `arm_64` [W/L/M] [getOSArchitecture]
- `chmod` in Files sections [L/M] [chmod]
- `importlib "uib_macosinstalllib"`
  - `install_macos_app($myapp$ : string) : string` [M]
  - `install_macos_pkg($mypkg$ : string) : string` [M]
  - `install_macos_dmg($mydmg$ : string) : string` [M]
  - `install_macos_zip($myzip$ : string) : string` [M]
  - `install_macos_generic($myfile$ : string) : string` [M]

The details to these library commands you will find here: Documentation of opsi library: uib_macosinstalllib.opsiscript

References to more macOS specific library functions you will find at: macOS specific functions

4.9. Example scripts for macOS
### 4.9.1. Run on macOS only

```plaintext
[Actions]
DefVar $OS$

set $OS$ = GetOS

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

### 4.9.2. Which macOS Version

The code:

```plaintext
Set $macosinfomap$ = getMacosVersionMap
```

gives (for example) the log:

```
The value of the variable "$macosinfomap$" is now:
(string 0)Release=11.0
(string 1)Build=20A3364e
(string 2)kernel name=Darwin
(string 3)node name=vmmac1100onmm1.uib.local
(string 4)kernel release=20.1.0
(string 5)kernel version=Darwin Kernel Version 20.1.0: Fri Aug 28 20:45:30 PDT 2020; root:xnu-7195.40.65.0.2~61/RELEASE_X86_64
(string 6)machine=x86_64
(string 7)processor=i386
(string 8)operating system=macOS
```
Chapter 5. Start and Command Line Options

Since version 4.11.3, the *opsi-script* program contains at Windows a manifest with the statement:

```xml
<requestedExecutionLevel level="highestAvailable" />
```

This means that if *opsi-script* is called on an NT6 OS by an Administrator, then it will run as an elevated process. If *opsi-script* is called with normal user privileges, then it will run with the privileges of this user.

If you start *opsi-script* without any parameter, it will start in the interactive mode.

*opsi-script* can be started with different sets of parameters depending on context and purpose of use.

**Note**

At Linux or macOS the parameter char is not "/" as here described for Windows but ".-". So instead of using *opsi-script* /help as we do on Windows, we use at Linux / macOS *opsi-script* -help.

Generic Options:

- `/?` or `/h[elp]`
  Show help

- `/silent`
  Run opsi-script without GUI

Execute one (or more) scripts:

```bash
opsi-script <scriptfile>[;<scriptfile>]* [<logfile>]
```

where:

- `<scriptfile>` = Name of the script file (incl. path).
- `<logfile>` = Name of the log file (incl. path). Paths to log files see also: Section 5.1, “Log File and Paths”

  - `/parameter <parameterstring>`
    A string that can be passed to the executed script and can be retrieved by the command *Paramstr*.
    Hereby is `<parameterstring>` a string without whitespaces.

  - `/logfile <logfile>`
    Define the log file:
    Hereby is:
    `<logfile>` = Name of the log file (incl. path). Paths to log files see also: Section 5.1, “Log File and Paths”

  - `/lang <lang>`
    Define the localization:
    Hereby is:
    `<lang>` = The two char language abbreviation (de,en,fr,es,…)

  - `/batch`
    Execute the given script with the batch GUI. The batch GUI has no possibility for user interaction. In combination with the option `/silent` there will be no GUI at all. If you call *opsi-script* without the option `/batch` the interactive GUI is started, which is designed for development and testing purposes.

  - `/productid <productId>`
    For the use together with `/servicebatch`; see overthere.

  - `/servicebatch`
    Execute the given script with the batch GUI and with a connection to the opsi web service. Thereby the given script will be executed as it would be if the opsi product given by the option `/productid` had the action request *setup*.
    The script file has to be the first option.
    The option `/opsiservice` and its sub options has to be given.
    The option `/productid` has to be given. This one is used for the communication with the opsi web service to run the given script as it would be the *setup-script* of the opsi product given with this option.

  - `/logproductid <productId>`
    While creating the log file the given `<productId>` should be used to note it as source of the log file.

  - `/normalwindow`
    Switches off the maximize if of the GUI in the not interactive mode.
• `/usercontext <[domain]\username>`
  If the given user is logged in then `opsi-script` will try to resolve the constants like `%CurrentAppdataDir%`, `%CurrentStartmenuDir%`, `%CurrentDesktopDir%`, `%CurrentStartupDir%`, `%CurrentProgramsDir%`, `%CurrentSendToDir%`, `%CurrentProfileDir%` from the context of the given user.
  Mostly used together with the 'User Profile Management' opsi extension.

• `/opsiservice <opsiserviceurl>`
  `/clientid <clientname>`
  `/username <username>`
  `/password <password>`
  `/sessionid <sessionid>`
  `/credentialfile <credentialfile>`
  Give the connection data to connect to the opsi web service:
  Hereby you have to give either `/clientid` and `/username` and `/password` and also optional the `/sessionid`
  or you have to give this data via a `/credentialfile`.

Declare what should be done in the context of the connected `/opsiservice`

• Default (none of the following parameters):
  Process the action requests as they stored for this client on the opsi-server.

• `/allloginscripts` or `/loginscripts`
  Process the login scripts of the opsi products. Using `/allloginscripts` all login scripts that are known to the opsi-server will be processed, no matter iftheses products are known to the client or not. Using `/loginscripts` only these login scripts will be precessed that belong to products that are installed or were installed and then removed (technical: there is an existing productOnClient object).

• `/productlist <productid>[,<productid>]*`
  Process the given `/productlist` in a way as it would normally done if there are the action request setup is stored at the opsi-server.
  Usually used by the `event_silent_install`.

• `/processproducts <productid>[,<productid>]*`
  Process the action requests as they stored for this client on the opsi-server but limited to the list of products given by `/processproducts`.

5.1. Log File and Paths

The default log file name is `opsi-script.log`. You may find (by default) up to 8 backup copies of old log files: from `opsi-script_0.log` until `opsi-script_8.log`.

The default number of backups may be overwritten by using the config: `[opsi-script-configs_log_rotation_count]`

The log file encoding is UTF-8.

By default log files are written at Windows into the directory `c:\opsi.org\log` which `opsi-script` tries to create. If `opsi-script` has no access to this directory it uses the user-TEMP directory.

At Linux: If running as `root` (default): `/var/log/opsi-script` If running as any other user: `/tmp`

The log file name and location will be overwritten via the specific command line option.

In the case, that `opsi-script` executes a script in `/batch` mode and with a specified (and working) usercontext, the default logging path is the `opsi\tmp` in the appdata directory of the user. This will be overwritten by an explicit given log path.

More possibilities to configure the logging:

see also: `[opsi-script-configs_writeProductLogFile]`
see also: `[opsi-script-configs_default_loglevel]`
see also: `[opsi-script-configs_force_min_loglevel]`
see also: `[opsi-script-configs_debug_lib]`
see also: `[opsi-script-configs_log_rotation_count]`

Beside the normal log file there is also a log file named `opsi-script.history`. This log file contains one line for every run of a product script since the first run. These lines have the pattern:

`<timestamp> handled: <productid> Version: <version> Request: <request> Result: <result>`
5.2. Central configuration via opsi Configs (Host Parameter)

Using opsi Configs (Host-Parameter) you may now change the logging:

- **opsi-script.global.debug_prog**: boolean
  If false log messages that are only relevant for debugging the opsi-script program itself are not written excepting Warnings and Errors. Default: false
  This will keep the log files smaller because you will find only messages that are relevant for understanding what your script is doing. The adjustment of all log messages to this new way is in progress and will take a while since all (about 1700) log calls inside the code are reviewed.

- **opsi-script.global.debug_lib**: boolean
  If false log messages from defined functions that are imported from external library files will be suppressed excepting Warnings and Errors. Default: false

- **opsi-script.global.default_loglevel**: intstr
  Sets (overrides) the default log level that is implemented inside the opsi-script code. This config has no effect on scripts where the loglevel is explicit set by a `setLogLevel` statement. Default: 6
  see also [SetLogLevel]
  see also [opsi-script-configs_force_min_loglevel]

- **opsi-script.global.force_min_loglevel**: intstr
  Forces a minimal log level.
  This can be used while debugging or development to set temporary and for selected clients a higher log level without changing the script. Default: 0
  see also [SetLogLevel]
  see also [opsi-script-configs_default_loglevel]

- **opsi-script.global.ScriptErrorMessages**: boolean
  This config overwrites the opsi-script internal default value for `ScriptErrorMessages` if opsi-script is running in the context of the opsi web service. If the value is true, syntactical errors trigger a pop up window with some informations on the error. This is in productive environments no good idea. Therefore the default value for this config is false. Inside a script the statement `ScriptErrorMessages` may be used to set this different from the defaults. Default: false
  see also: [ScriptErrorMessages]

- **opsi-script.global.AutoActivityDisplay**: boolean
  If true shows a marquee (endless) progressbar while external processes (winbatch/dosbatch sections) are running. Default: true
  see also: [AutoActivityDisplay]
* opsi-script.global.SupressSystemEncodingWarning: boolean
  If true the warning: Encoding=system makes the opsiscript not portable between different OS will be supressed.
  Default: false
  see also: [encoding]

* opsi-script.global.ReverseProductOrderByUninstall: boolean
  If true the product list is reordered so that uninstall actions will be conducted first and in reverse order as the products were installed
  Default: false

* opsi-script.global.log_rotation_count: string (number) // seit 4.12.4.29
  Defines the number of opsi-script.log backups that will be stored in the log directory on the client. (opsi-script_0.log, opsi-script_1.log, ...)
  Default = 8 ; Max = 999

* opsi-script.global.writeProductLogFile: boolean // seit 4.12.4.35
  If true logs for every single opsi product will be created. You will find these log in the sub directory lastprodlogs of the opsi-script log directory (E.g. c:\opsi.org\log\lastprodlogs). The name of these log files is <productId>.log. There is always only the last log for every product. If the product script contains reboots in (not after) the script, the log contains only the part after the last reboot.
  Default: false
Chapter 6. Additional Configurations

6.1. Central Logging of Error Messages

If the *opsi-script* running in opsi web service mode, it sends the log file via opsi web service to the opsi server.

6.2. Skinnable *opsi-script* [W/L/M]

Since version 3.6 the *opsi-script* GUI can be customized. The elements for customizing are to be found in the winskin subdirectory of the *opsi-script* directory. The configuration file for customization is skin.ini.

Since version 4.12.4.15 the *opsi-script* searches the skin directory in the following order (directory with the first skin.ini to be found wins):

**Windows:**

%OpsiScriptDir% = C:\Program Files (X86)\opsi.org\opsi-client-agent\opsi-script

1. %OpsiScriptDir%\..\custom\customskin
2. %OpsiScriptDir%\skin
3. %OpsiScriptDir%\winstskin (for backward compatibility)

**Linux:**

%OpsiScriptDir% = /opt/opsi-script

1. /usr/share/opsi-script/skin
2. /usr/share/opsi-script/customskin (for backward compatibility)
3. %OpsiScriptDir%\skin

**macOS:**

%OpsiScriptDir% = /Applications/opsi-script/Contents/macOS

1. /usr/share/opsi-script/skin
2. %OpsiScriptDir%/../Resources/skin

see also: [opsi-script-linux-path](#)

see also: [opsi-script-macos-path](#)

With the Command `SetSkinDirectory` the SkinDirectory to be used can be defined in the script. If the path specified is empty or not valid, the default path will be used.

Example:

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

6.3. *opsi-script* encoding [W/L/M]

Some technical notes about the wording:

- ASCII, plain ASCII
ASCII stands for: American Standard Code for Information Interchange
plain ascii: 7 Bit / per char for 128 different chars. This will be found as base of the following.

- **ANSI, Codepages**
  Using 8 Bit (Byte) / per char for 255 different chars. Lower 128 chars = ASCII, Upper 128 Chars according to selected Code page. Well known code pages:
  Windows-1252 = CP1252 = ISO 8851-1 = Western Europe code page.
  The first 256 Chars of CP1252 are also part of Unicode.
  ANSI stands for American National Standards Institute:
  https://stackoverflow.com/questions/701882/what-is-ansi-format
  ANSI encoding is a slightly generic term used to refer to the standard code page on a system, ( ... )The name ”ANSI” is a misnomer, since it does not correspond to any actual ANSI standard, but the name has stuck.
  So what is an an ANSI-String?
  https://wiki.freepascal.org/Character_and_string_types#AnsiChar says:
  A variable of type AnsiChar, also referred to as char, is exactly 1 byte in size, and contains one ”ANSI” (local code page) character.
  The problems with using code pages are:
  - You have to use different encoding in different places of the world.
  - Maximum 255 Chars can be coded, but some languages have a lot more characters.

- **Unicode, UTF-8**
  Unicode is (like ANSI) a encoding family (and not a encoding). The most important difference to using code pages is, that here we use to encode one char (possibly) more than one byte (exactly: up to 4 bytes).
  The most important members of the Unicode family are:
  - UTF-16-LE (also some times called Windows Unicode):
    Uses a minimum length of 2 Bytes per char up to 4 Bytes. The LE stands for Little Endian and tell us that the most significant byte of a char is the last one. (char n : LE=6E 00, BE=00 6E)
  - UTF-8:
    Uses for the chars out of plain ASCII one byte but for every thing else 2 up to 4 bytes.
    So in fact: a file that uses only plain ASCII is the same binary wether you save it in cp1252 or utf8.
  - BOM
    A file with Unicode encoding may contain in the first 4 Bytes the information about the used (unicode-)encoding. This is the BOM (Byte Order Mark). If there is a BOM, opsi-script will detect and use it.

The default encoding for a script is the encoding of the running operating system. So for example one script will be interpreted on a Greek windows system as encoded with cp1253 on a German windows system as cp1252 and under Linux as UTF-8.

We strongly recommend to create all your opsiscript files in UTF-8 encoding and add the line `encoding=utf8` to the file.
This makes your files portabel.
See also below.

- **encoding=<encoding>**
  Since Version 4.11.4.1 it is possible to define the encoding in the script. This may be done in the main script and in the sub scripts, includes and libraries as well. You have to give the command:
  `encoding=<encoding>`
  This command can be at any position in the code (Even before `[actions]`).
  If the command `encoding=` is missing, than the expected encoding is the system encoding of the running operating system. At Linux and macOS this would be UTF-8. At Windows the system encoding a cp* and depends on the localization. In Western Europe for example z.B. cp1252.
  If the input file contains umlauts (is not pure plain ASCII) and there is no line: `encoding=utf8`, then you will get a warning:
  `Encoding=system makes the opsiscript not portable between different OS`.
  This Warning may be suppressed by the config (Host Parameter):
  `opsi-script.global.supresssystemencodingwarning = true`.
  siehe auch `opsi-script-configs_supresssystemencodingwarning`

  If the input file contains umlauts (is not pure plain ASCII) and there is a discrepancy between the detected encoding (c.f. by a BOM) and the implicit encoding `system` or the given encoding by `encoding=`, then you will get the following warning:
  `Warning: Given encodingString <> is different from the expected encoding <>`
Using the command: `encoding=<encoding>`
the `<encoding>` may be one of the following values:

<table>
<thead>
<tr>
<th>encoding</th>
<th>allowed alias</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td></td>
<td>use the encoding of the running OS</td>
</tr>
<tr>
<td>auto</td>
<td></td>
<td>try to guess the encoding</td>
</tr>
<tr>
<td>UTF-8</td>
<td>utf8</td>
<td></td>
</tr>
<tr>
<td>UTF-8BOM</td>
<td>utf8bom</td>
<td></td>
</tr>
<tr>
<td>Ansi</td>
<td>ansi</td>
<td>8 Bit encoding with Codepage</td>
</tr>
<tr>
<td>CP1250</td>
<td>cp1250</td>
<td>Central and East European Latin</td>
</tr>
<tr>
<td>CP1251</td>
<td>cp1251</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>CP1252</td>
<td>cp1252</td>
<td>West European Latin</td>
</tr>
<tr>
<td>CP1253</td>
<td>cp1253</td>
<td>Greek</td>
</tr>
<tr>
<td>CP1254</td>
<td>cp1254</td>
<td>Turkish</td>
</tr>
<tr>
<td>CP1255</td>
<td>cp1255</td>
<td>Hebrew</td>
</tr>
<tr>
<td>CP1256</td>
<td>cp1256</td>
<td>Arabic</td>
</tr>
<tr>
<td>CP1257</td>
<td>cp1257</td>
<td>Baltic</td>
</tr>
<tr>
<td>CP1258</td>
<td>cp1258</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>CP437</td>
<td>cp437</td>
<td>Original IBM PC hardware code page</td>
</tr>
<tr>
<td>CP850</td>
<td>cp850</td>
<td>&quot;Multilingual (Latin-1)&quot; (Western European languages)</td>
</tr>
<tr>
<td>CP852</td>
<td>cp852</td>
<td>&quot;Slavic (Latin-2)&quot; (Central and Eastern European languages)</td>
</tr>
<tr>
<td>CP866</td>
<td>cp866</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>CP874</td>
<td>cp874</td>
<td>Thai</td>
</tr>
<tr>
<td>CP932</td>
<td>cp932</td>
<td>Japanese (DBCS)</td>
</tr>
<tr>
<td>CP936</td>
<td>cp936</td>
<td>GBK Supports Simplified Chinese (DBCS)</td>
</tr>
<tr>
<td>CP949</td>
<td>cp949</td>
<td>Supports Korean (DBCS)</td>
</tr>
<tr>
<td>CP950</td>
<td>cp950</td>
<td>Supports Traditional Chinese (DBCS)</td>
</tr>
<tr>
<td>ISO-8859-1</td>
<td>iso8859-1</td>
<td>Latin-1</td>
</tr>
<tr>
<td>ISO-8859-2</td>
<td>iso8859-2</td>
<td>Latin-2</td>
</tr>
<tr>
<td>KOI-8</td>
<td>koi8</td>
<td>Kyrillisches Alphabet</td>
</tr>
<tr>
<td>UCS-2LE</td>
<td>ucs2le</td>
<td>(UTF-16-LE, Windows Unicode Standard)</td>
</tr>
<tr>
<td>UCS-2BE</td>
<td>ucs2be</td>
<td>(UTF-16-BE)</td>
</tr>
</tbody>
</table>
Sources see:


https://en.wikipedia.org/wiki/UTF-8
Chapter 7. The \textit{opsi-script} Script

On principle: \textit{opsi-script} is an interpreter for a specific, easy to use scripting language which is tailored for the requirements of software installations. A script should be an integrated description, and a means of control, for the installation of one piece of software.

The following section sketches the structure of a \textit{opsi-script} script. The purpose is to identify the book marks of a script: in which way we have to look into it, to understand its processing.

All elements shall be described more in detail in the further section. The purpose then will be to show how scripts can be modified or developed.

7.1. An Example

\textit{opsi-script} scripts are roughly derived from .INI files. They are composed of sections, which are marked by a title (the section name) which is written in brackets [].

Schematically a \textit{opsi-script} script looks like this one (here with a check which operating system is installed):

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

;which Windows-Version?
DefVar $MSVersion$
Set $MSVersion$ = GetMsVersionInfo
if CompareDotSeparatedNumbers($MSVersion$,"\>\=\","6")
    sub_install_winnt6
else
    stop "not a supported OS-Version"
endif

[sub_install_winnt6]
Files_copy_winnt6
WinBatch_Setup

[Files_copy_winnt6]
copy "%scriptpath%\files_win10\*.*" "c:\temp\installation"

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

How can we read the sections of this script?

7.2. Primary and Secondary Subprograms of a \textit{opsi-script} script

The script as a whole serves as a program, an instruction for an installation process. Therefore each of its sections can be seen as a subprogram (or "procedure" or "method"). The script is a collection of subprograms.

The human reader as well as an interpreting software has to know at which element in this collection reading must start.

Execution of a \textit{opsi-script} script begins with working on the [Actions] section. All other sections are called as subroutines. This process is only recursive for Sub sections: Sub sections have the same syntax as Actions sections and may contain calls for further subroutines.

If a script is run as \texttt{userLoginScript} and it contains a section [ProfileActions], so the script interpretation will be started at the \texttt{ProfileActions} section.

This gives reason to make the distinction between primary and secondary subprograms:

The primary or general control sections comprise

- the \texttt{Actions} section
- \texttt{Sub} sections (0 to n subroutines called by the \texttt{Actions} section which are syntactical and logical extensions of the calling section).
the ProfileActions section, which will be interpreted in different ways according to the script mode (Machine/Login).

The procedural logic of the script is determined by the sequence of calls in these sections.

The secondary or specific sections can be called from any primary section but have a different syntax. The syntax is derived from the functional requirements and library conditions and conventions for the specific purposes. Therefore no further section can be called from a secondary section.

At this moment there are the following types of secondary sections:

- Files sections,
- WinBatch sections,
- DosBatch/DosInAnIcon/ShellInAnIcon sections,
- Registry sections
- Patches sections,
- PatchHosts sections,
- PatchTextFile sections,
- XMLPatch sections (discouraged),
- XML2 sections,
- LinkFolder sections,
- opsiServiceCall sections,
- ExecPython sections,
- ExecWith sections,
- LDAPsearch sections.

Meaning and syntax of the different section types are treated in Syntax and Meaning of Primary Sections of a opsi-script Script and Secondary Sections.

7.3. String Expressions in a opsi-script Script

Textual values (string values) in the primary sections can be given in different ways:

- A value can be directly cited, mostly by writing in into (double) citation marks. Examples:
  "Installation of Mozilla"
  "n:\home\user name"

- A value can be given by a String variable or a String constant, that "contains" the value:
  The variable $MsVersion$ may stand for "6.1" – if it has been assigned beforehand with this value.

- A function retrieves or calculates a value by some internal procedure. E. g. EnvVar ("Username")
  fetches a value from the system environment, in this case the value of the environment variable Username. Functions may have any
  number of parameters, including zero:
  GetMsVersionInfo
  On a win7 system, this function call yields the value "6.1" (not as with a variable this values has to be produced at every call again).

- A value can be constructed by an additive expression, where string values and partial expressions are concatenated - theoretically "plus"
  can be seen as a function of two parameters:
  $Home$ + "mail"

(More on this in String Expressions, String Values, and String Functions)

There is no analogous way of using string expressions in the secondary sections. They follow there domain specific syntax. e.g. for copying
commands similar to the windows command line copy command. Up to this moment it is no escape syntax implemented for transporting
primary section logic into secondary sections.

The only way to transport string values into secondary sections is the use of the names of variables and constants as value container in these
sections. Let's have a closer look at the variables and constants of an *opsi-script* script:
Chapter 8. Definition and Use of Variables and Constants in a *opsi-script* Script

8.1. General

In a *opsi-script* script, variables and constants appear as "words", that are interpreted by *opsi-script* and "contain" values. "Words" are sequences of characters consisting of letters, numbers and some special characters (in particular ".", ",", "="), but not blanks, but no brackets, parentheses, or operator signs ("+"). *opsi-script* variables and constants are not case-sensitive.

There exist the following types of variables or constants:

- **Global text constants**, shortly constants, contain values which are present by the *opsi-script* program and cannot be changed in a script. Before interpreting the script *opsi-script* replaces each occurrence of the pure constant name with its value in the whole script (textual substitution).
  
  An example will make this clear:
  
  The constant `%ScriptPath%` is the predefined name of the location where *opsi-script* found and read the script that it just executes. This location may be, e.g., `p:\product`. Then we have to write `%ScriptPath%` in the script when we want do get the value "p:\product".

  - observe the citations marks which include the constant delimiter.

- **Text or String variables**, shortly variables, have an appearance very much like any (String) variables in a common programming language. They must be declared by a `DefVar` statement before they can be used. In primary sections, values can be assigned to variables (once ore more times). They can be used as elements in composed expressions (like addition of strings) or as function arguments.

  But they freeze in a secondary section to a phenomenon that behaves like a constant. There, they appear as a non-syntactical foreign element. Their value is fixed and is inserted by textual substitution for their pure names (when a section is called, whereas the textual substitution for real constants take place before starting the execution of the whole script).

- **Stringlist variables** are declared by a `DefStringList` statement. In primary sections they can be used for many purposes, e.g. collecting strings, manipulating strings, building sections.

In detail:

8.2. Global Text Constants

Scripts shall work in a different contexts without manual changes. The contexts can be characterized by system values as OS version or certain paths. *opsi-script* introduces such values as constants into the script.

8.2.1. Usage

The fundamental characteristics of a text constant is the way how the values which it represents come intro the script interpretation process:

The name of the constant, that is the pure sequences of chars, is substituted by its fixed value in the whole script before starting the script execution.

The replacement does not take into account any syntactical context in which the name possibly occur (exactly like with variables in secondary sections).

8.2.2. Example

*opsi-script* implements constants `%ScriptPath%` for the location of the momentarily interpreted script and `%System%` for the name of the windows system directory. The following (Files) subsection defines a command that copies all files from the script directory to the windows
system directory:

```plaintext
files_do_my_copying

\copy \%ScriptPath%\system\*.* \%System%
```

At this moment the following constants are implemented:

### 8.2.3. System paths

#### Base system directories [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:\Users`

#### Common (All Users) directories [W]

- `%AllUsersProfileDir% or %CommonProfileDir%: C:\Users\Public`
- `%CommonStartMenuPath% or %CommonStartmenuDir%: C:\ProgramData\Microsoft\Windows\Start Menu`
- `%CommonAppdataDir%: C:\ProgramData`
- `%CommonDesktopDir%`
- `%CommonStartupDir%`
- `%CommonProgramsDir%`

<table>
<thead>
<tr>
<th>Constant</th>
<th>Win7 - Win10 (NT6)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%AllUsersProfileDir%</code></td>
<td>C:\Users\Public</td>
</tr>
<tr>
<td><code>%CommonProfileDir%</code></td>
<td>C:\Users\Public</td>
</tr>
<tr>
<td><code>%CommonStartMenuPath%</code></td>
<td>C:\ProgramData\Microsoft\Windows\Start Menu</td>
</tr>
<tr>
<td><code>%CommonAppDataDir%</code></td>
<td>C:\ProgramData</td>
</tr>
<tr>
<td><code>%CommonDesktopDir%</code></td>
<td>C:\Users\Public\Desktop</td>
</tr>
<tr>
<td><code>%CommonStartupDir%</code></td>
<td>C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup</td>
</tr>
<tr>
<td><code>%CommonProgramsDir%</code></td>
<td>C:\ProgramData\Microsoft\Windows\Start Menu\Programs</td>
</tr>
<tr>
<td><code>%AllUsersProfileDir%</code></td>
<td>C:\Users\Public</td>
</tr>
<tr>
<td><code>%DefaultUserProfileDir%</code></td>
<td>C:\Users\Default</td>
</tr>
</tbody>
</table>
Default User Directory [W]

%DefaultUserProfileDir%

Current (logged in or usercontext) user directories [W]

%AppdataDir% or %CurrentAppdataDir%: // since 4.10.8.13
NT6: c:\users\%USERNAME%\Appdata\Roaming

%CurrentStartmenuDir%
%CurrentDesktopDir%
%CurrentStartupDir%
%CurrentProgramsDir%
%CurrentSendToDir%
%CurrentProfileDir% // since 4.11.2.1

/AllUserProfiles (/AllNTUserProfiles) directory constants [W/L/M]

In Files sections that are called with option /AllUserProfiles there is a pseudo variable
%UserProfileDir%
When the section is executed for each user that exists on a work station this variable represents the name of the profile directory of the user just treated.
The parameter /AllUserProfiles exits since 4.12.4.27. The use of the older and still working synonym /AllNTUserProfiles is discouraged.

%CurrentProfileDir% // since 4.11.2.1
may be used instead of the older %UserProfileDir% in order to have Files-sections which may be used also for userLoginScripts.

%UserProfileDir% or %CurrentProfileDir%
NT6: c:\users\%USERNAME%

8.2.4. opsi-script Path and Directory [W/L/M]

%ScriptPath% or %ScriptDir%: represents the path of the current opsi-script script (without closing backslash). Using this variable we can build path and file names in scripts that are relative to the location of the script. So, everything can be copied, called from the new place, and all works as before.

%RealScriptPath%: If the script is called via symlink then it is the resolved version of %scriptpath% else it is the same as %ScriptPath% (since 4.12.4.21)

%ScriptDrive%: The drive where the just executed opsi-script script is located (including the colon).

%OpsiScriptDir% (since 4.12.3.6)
The location (without closing backslash) of the running opsi-script.
Identical with the outdated form: %WinstDir%

%OpsiScriptVersion% (since 4.12.3.6)
Version string of the running \textit{opsi-script}.
Identic with the outdated form: \%WinstVersion\%

\%opsiscriptProcname\%  (since 4.12.4.35)
Name of the running \textit{opsi-script} process.
Can be used in combination with \texttt{isProcessChildOf}

\%Logfile\%: The name of the logfile which \textit{opsi-script} is using.

\%opsiTmpDir\%  // since 4.11.4.3
Directory which should be used for temporary files. (At Windows: c:\opsi.org\tmp)

\%opsiUserTmpDir\%  // since 4.12.4.37
Directory which should be used for temporary files and for which you do not need administrator privileges. (At Windows: c:\opsi.org\usertmp)

\%opsilogDir\%  // since 4.11.4.3
Directory which should be used for log files. (At Windows: c:\opsi.org\log)

\%opsiScriptHelperPath\%
Corresponds to: \%ProgramFiles32Dir\%\opsi.org\opsiScriptHelper
Path in which the help program, libraries, and if items needed for script execution could be installed.
Since 4.11.3.2

\%opsidata\%  // since 4.12.0.12
Directory which should be used for opsi data files (e.g. disks, partitions). (At Windows: c:\opsi.org\data)

\%opsiapplog\%  // since 4.12.0.12
Directory which should be used for log files from programs that running in the user context. (At Windows: c:\opsi.org\applog)

Example:
The code:

\begin{verbatim}
message "Testing constants: " + $\%OpsiscriptVersion\% + 
set $ConstTest$ = $\%OpsiscriptVersion\%$
if $\%OS\%$ = "Windows_NT"
set $InterestingFile$ = \%opsiscriptdir\%\opsi-script.exe
if not (FileExists($InterestingFile$))
  set $InterestingFile$ = \%opsiscriptdir\%\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
\end{verbatim}

results to the following log:
### 8.2.5. Network Information [W/L/M]

**%Host%**: (discouraged) The value of a environmental variable host (traditionally meaning the opsi server name, not to confuse with %HostID% (meaning the client network name)).

**%PCName%**: The value of the environmental variable PCName, when existing. Otherwise the value of the environmental variable computername. (Should be the netbios name of the PC)

**%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%**: (discouraged) may be the IP-Address of the machine. Use function `GetMyIpByTarget()` instead. see also: [GetMyIpByTarget]

**%Username%**: Name of the logged in user.

### 8.2.6. Data for and from opsi service [W/L/M]

**%HostID%**: Should be the fully qualified domain name of the opsi client as it is supplied from the command line or otherwise. If running in opsi service context it is better to use %opsiserviceUser%.

**%opsiserviceURL%**: The (usually https://) URL of the opsi service.(https://<opsiserver>:4447)

**%opsiserver%**: The server name derived from the %opsiserviceURL%.

**%opsidepotId%**: Depot Server (FQDN) //since 4.11.4
%opsiserviceUser%: The user ID for which there is a connection to the opsi service. If running in opsi service context this is usually the client FQDN used by opsi.

%opsiserviceClientId%: At start of opsi-script in opsi service context it is set to the command line option /clientid. So it will be the FQDN of the client in opsi service context and it will be empty without opsi service context.

%opsiservicePassword%: The user password used for the connection to the opsi service. The password is eliminated when logging by the standard opsi-script logging functions.

%installingProdName%: The productid of the product that is actually installed via call by the opsi-service. Empty if the Script ist not started by the opsi-service.

%installingProdVersion%: A String combined from <productversion>-<packageversion> for the product that is actually installed via call by the opsi-service. Empty if the Script ist not started by the opsi-service.

%installingProduct%: (discouraged) The name (productId) of the product for which the service has called the running script. In case that there the script is not run via the service the String is empty.

8.3. String (or Text) Variables [W/L/M]

8.3.1. Declaration

String variables must be declared before they can be used. Since 4.12.4.32 it is also possible to pass an optional initial value. The syntax for the declaration reads

DefVar <variable name> [= <initial value>]

e.g.

DefVar $MsVersion$ ; since 4.12.4.32 also possible:
DefVar $MsVersion$ = '10.0'

Explanation:

• Variable names do not necessarily start or end with a dollar sign, but this is strongly recommended as a convention to avoid problems by the replacement of variable names by their value in secondary sections.

• Variables can only be declared in primary sections (Actions section, sub sections and ProfileActions).

• The declaration should not depend on a condition. That is it should not placed into a branch of an if – else statement. Otherwise, it could happen that the DefVar statement is not executed for a variable, but an evaluation of the variable is tried in some if clause (such producing a syntax error).

• The variables are initialized with an empty string (""").

Recommendation:

• The first and last letter of the name should be $.

• Define all variables at the beginning of the script.

8.3.2. Value Assignment

As it is appropriate for a variable, it can take on one value resp. a series of values while a script is progressing. The values are assigned by statements with syntax

Set <Variablename> = <Value>

<Value> means any (String valued) expression.
Examples (For Examples see String Expressions, String Values, and String Functions):

Set $OS$ = GetOS
Set $WinVersion$ = "unknown"

if $OS$ = "Windows_NT"
  Set $WinVersion$ = GetMsVersionInfo
endif

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

8.3.3. Use of variables in String expressions

In primary sections of an Opsi-Script script, a variable "holds" a value. When it is declared it is initialized with the empty String "". When a new value is assigned to it via the set command, it represents this value.

In a primary section a variable can replace any String expression resp. can be a component of a String expression, e.g.

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

In a primary section the variable name denotes an object that represents a string. If we add the variable we mean that the underlying string shall be added somehow.

This representational chain is shortcut in a secondary section. Just the variable name now stands for the string.

8.3.4. Secondary vs. primary sections

When a secondary section is loaded and Opsi-Script starts its interpretation the sequence of chars of a variable name is directly replaced by the value of the variable.

Example:
A copy command in a files section shall copy a file to 
"n:\home\user name\mail\backup" kopiert werden.

We first set $MailLocation$ to the directory above it:

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

$MailLocation$ is now holding
"n:\home\user name\mail"

In a primary section we may now express the directory 
"n:\home\user name\mail\backup" by 
$MailLocation$ + "\backup"

The same directory has to be designated in a secondary section as:
"$MailLocation$\backup"

A fundamental difference between the thinking of variables in primary vs. secondary sections is that, in a primary section, we can form an assignment expression like 
$MailLocation$ = $MailLocation$ + "\backup"

As usual, this means that $MailLocation$ first has some initial value and takes on a new value by adding some string to the initial value. The reference from the variable is dynamic, and may have a history.
In a secondary section any such expression would be worthless (and eventually wrong), since `$MailLocation$` is bound to be replaced by some fixed string (at all occurrences virtually in the same moment).

### 8.4. Stringlist Variables [W/L/M]

Variables for string lists must be declared in a `DefStringList` statement. Since 4.12.4.32 it is also possible to pass an optional initial value. The syntax for the declaration reads

```
DefStringList <VarName> [= <initial value>]
```

A string list can serve e.g. as container for the captured output of a shell program. The collected strings can be manipulated in a lot of ways. In detail this will be treated in the section on string list processing (see `String List Functions and String List Processing`).
Chapter 9. Syntax and Meaning of Primary Sections of a opsiscrypt Script [W/L/M]

As shortly presented in chapter Primary and Secondary Subprograms of a opsiscrypt script the Actions section of a script can be regarded as a the main method of the opsiscrypt script and describes the global processing sequence. It may call subroutines - the Sub sections which may then recursively call Sub sections themselves.

The following sections explain syntax and use of the primary sections of a opsiscrypt script.

9.1. Primary Sections [W/L/M]

There are possibly three kinds of primary sections in a script

- an Initial section (may be omitted),
- an Action section,
- any number of Sub sections
- an ProfileActions section

Initial and Action section are syntactically equivalent (but Initial has to keep the first place). By convention, in the Initial section some parametrizations of the script execution (e.g. the loglevel) are made. The Action section can be regarded as the main program in a opsiscrypt script. It contains the sequence of actions that are controlled by the script.

Sub sections are as well syntactically equivalent. But they are a called from the Action section. Then, they can call themselves Sub sections.

A Sub section is determined by creating a name that begins with "Sub", e.g. Sub_InstallBrowser. By writing its name in the Action section we produce a call to the Sub section. The meaning of this call is defined by the content of the section in the script that begins with the bracketed name, in the example [Sub_InstallBrowser]

Subsections of second and higher order cannot host internal sections. Instead, their procedure calls must refer to sections defined in the main script file or defined as external sections (cf. Subprogram Calls).

If (nested) sub sections are externalized to external files, the called sections has to be in that file where they are called from. According to the complexity of the script they may sometimes have to be placed also in the main file.

A ProfileActions section at a normal installation script may be used as a sub section with a special syntax. In a userLoginScript this section will be used as script start (instead of Actions). See chapter User Profile Management at the opsimanual and Commands for userLoginScripts / User Profile Management.

9.2. Parametrizing opsiscrypt [W/L/M]

9.2.1. Specification of Logging Level [W/L/M]

The old function LogLevel= is discouraged since opsiscrypt version 4.10.3. For backward compatibility reasons Loglevels ste by this old function will be increased by 4 before they are used.

There are two syntactical variants for specifying the logging level:

- SetLogLevel = <number>
- SetLogLevel = <String expression>
I.e. the number can be given as an integer value or as a string expression (cf. String Expressions, String Values, and String Functions). In the second case, *opsi-script* tries to evaluate the string expression as a number. There exist ten levels from 0 up to 9.

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nothing (absolute nothing)</td>
</tr>
<tr>
<td>1</td>
<td>essential (<em>essential information</em>)</td>
</tr>
<tr>
<td>2</td>
<td>critical (unexpected errors that may cause a program abort)</td>
</tr>
<tr>
<td>3</td>
<td>error (Errors that don’t will abort the running program)</td>
</tr>
<tr>
<td>4</td>
<td>warning (you should have a look at this)</td>
</tr>
<tr>
<td>5</td>
<td>notice (important statements to the program flow)</td>
</tr>
<tr>
<td>6</td>
<td>info (Additional Infos)</td>
</tr>
<tr>
<td>7</td>
<td>debug (important debug messages)</td>
</tr>
<tr>
<td>8</td>
<td>debug2 (a lot more debug informations and data)</td>
</tr>
<tr>
<td>9</td>
<td>confidential (passwords and other security relevant data)</td>
</tr>
</tbody>
</table>

The logging at the different log levels is:

- Log level 5: comments, messages, Execution of sections
- Log level 6: Statements, New values for stringvars, results of complete boolean expression
- Log level 7: new values for stringlist vars, output from external processes (shellInAnIcon) if the output is not assigned to a stringlist variable, results of parts of a boolean expression
- Log level 8: other stringlist output eg. string lists from stringlist functions and output from external processes (shellInAnIcon) that is assigned to a stringlist variable.

The default value is “7”. see also: [opsi-script-configs_default_loglevel] see also: [opsi-script-configs_force_min_loglevel]

### 9.2.2. Required opsi-script Version [W/L/M]

The statement

\[
\text{requiredOpsiscriptVersion} \text{<RELATION SYMBOL>} \text{<NUMBER STRING>}
\]

since 4.12.3.6

e.g.

\[
\text{requiredOpsiscriptVersion} \geq "4.12.3.6"
\]

makes *opsi-script* check if the desired version state is given. Otherwise an error message window pops up or is written to the log and the script result is failed.

There is no default.

A identical but outdated version is (since 4.3):

\[
\text{requiredWinEstVersion} \text{<RELATIONSSYMBOL>} \text{<ZAHLENSTRING>}
\]

### 9.2.3. Reacting on Errors [W/L/M]

There are two kinds of errors which are treated in different ways:

1. illegal statements which cannot be interpreted by *opsi-script* (syntactical errors),
2. failing statements which cannot be executed because of external, objective reasons (execution errors).
In principal, syntactical errors lead to a *script is failed* situation, execution errors are logged in a log file to be analysed later.

The behaviour of *opsi-script* when it recognizes a syntactical error is defined by the configuration statement

- **ScriptErrorMessages** = <boolean value>
  
  If the value is true (default), syntactical errors trigger a pop up window with some informations on the error. The boolean value may be *true* or *false*. Delimiters *on* or *off* can be used as well.
  
  Default=true
  
  see also: [opsi-script-configs_ScriptErrorMessages]

- **FatalOnSyntaxError** = <boolean value>
  
  - *true* = (default) If a syntax error occurs, then the script execution will be stopped and the script result will be set to *failed*. Also, the message *Syntax Error* will be passed to the opsi-server.
  
  - *false* = If a syntax error occurs, then the script execution will not be stopped and the script result will be set to *success*.

  In either case above, the syntax error will be logged as *Critical*.
  
  In either case above, the error counter will be increased by 1.
  
  Since 4.11.3.2
  
  In older versions there was no logging of syntax errors, no increase of error counter, and the result was always set to *success*.

- **FatalOnRuntimeError** = <boolean value>
  
  A Runtime Error is an script logic error that leads to an forbidden or impossible operation. An Example: You try to get the 5th string from a string list that have only 3 elements.
  
  - *true* = If a runtime error occurs, then the script execution will be stopped and the script result will be set to *failed*. Also, the message *Runtime Error* will be passed to the opsi-server.
  
  - *false* = (default) If a runtime error occurs, then the script execution will not be stopped and the script result will be set to *success*.
  
    The runtime error will be logged as *Error* and the error counter will be increased by 1.
  
    Since 4.11.4.3

  There two configuration options for execution errors.

- **ExitOnError** = <boolean value>
  
  This statement defines if the script execution shall terminate when an error occurs. If the value is true or yes the program will stop execution, otherwise errors are just logged (default).

- **TraceMode** = <boolean value>
  
  In TraceMode (default false) every log file entry will additionally be shown in message window with an O.K. button.

### 9.2.4. Staying On Top [W]

- **StayOnTop** = <Wahrheitswert>
  
  With StayOnTop = true (or = on) we request, that - in batch mode - the *opsi-script* window be on top on the windows which share the screen. That means it should be visible in the "foreground" as long as no other window having the same status wins.

  According to the system manual the value cannot be changed while the program is running. But it seems that we can give a new value to it once.

  *StayOnTop* has default false in order to avoid that some other process raises an error message which eventually can not be seen if *opsi-script* keeps staying on top.
9.2.5. Show window mode / Skin / Activity [W/L/M]

- **SetSkinDirectory** `<skindir>` // [W/L/M]
  Sets the skin directory to use and loads the skin. If this command is used with an empty or invalid path, the default skin dir is used. The default skin dir `%OpsiScriptDir%\skin`.

Example:
```plaintext
SetSkinDirectory "%ScriptPath%\testskin"
sleepseconds 1
SetSkinDirectory ""
```

see also: Skinnable opsi-script

To change the modes of how the opsi-script window is displayed, use these commands:

- **NormalizeWinst**
  Sets the opsi-script window to the normal mode

- **IconizeWinst**
  Sets the opsi-script window to the minimized mode

- **MaximizeWinst** //since 4.11.5
  Sets the opsi-script window to the maximized mode

- **RestoreWinst**
  Sets the opsi-script window to the mode before the last change

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

**forceLogInAppenMode** = <boolean value> //since 4.12.3.6 (default=false); if true log will be send in append mode

9.3. String Expressions, String Values, and String Functions [W/L/M]

A String expression can be

- an elementary String value
- a nested String value
- a String variable
- the concatenation of other String expressions
9.3.1. Elementary String Values

An elementary String value is any sequence of characters that is enclosed in double or single citations marks, formally:

"<sequence of characters>"

or

'<sequence of characters>'

Example:

```euclid
DefVar $ExampleString$
Set $ExampleString$ = "my Text"
```

9.3.2. Strings in Strings (Nested String Values)

If the sequence of chars itself contains citation marks we have to use the other kind of citation marks to enclose it:

```euclid
DefVar $citation$
Set $citation$ = 'he said "Yes"'
```

If the sequence of chars is containing both kinds of citation marks we must use the following special expression: 

**EscapeString**: <sequence of characters>

E.g. we can write:

```euclid
DefVar $Meta_citation$
Set $Meta_citation$ = EscapeString: Set $citation$ = 'he said "Yes"'
```

Then the variable $Meta_citation$ will exactly contain the complete sequence of chars that follows the colon after "EscapeString" (including the blank). Such, $Meta_citation$ will contain the complete statement:

```euclid
Set $citation$ = 'he said "Yes"
```

9.3.3. String Concatenation

String concatenation is written using the addition sign ("+")

<String expression> + <String expression>

Example:

```euclid
DefVar $String1$
DefVar $String2$
DefVar $String3$
DefVar $String4$
Set $String1$ = "my text"
Set $String2$ = "and"
Set $String3$ = "your text"
Set $String4$ = $String1$ + " " + $String2$ + " " + $String3$
```

$String4$ then has value "my text and your text".

9.3.4. String Variables

A String variable in a primary section "contains" a String value. In an String expression, it can always substitute an elementary string. For
how to define and set String variables cf. String (or Text) Variables.

The following sections present the variety of string functions.

### 9.3.5. String Functions which Return the OS Type

- **GetOS : string [W/L/M]**
  The function tells which type of operating system is running.
  GetOS returns one of the following values:
  - "Windows_NT" (including Windows 2000 to Windows 10)
  - "Linux"
  - "macos"

- **GetNtVersion [W]**
  Deprecated - please use **GetMSVersionInfo**.
  A Windows NT operating system is characterized by a the Windows type number and a subtype number. GetNtVersion returns the precise subtype name. Possible values are
  - "NT3"
  - "NT4"
  - "Win2k" (Windows 5.0)
  - "WinXP" (Windows 5.1)
  - "Windows Vista" (Windows 6)
  If the NT operating system has higher versions as 6 or there are version not explicitly known the function returns "Win NT" and the complete version number (5.2, … resp. 6.0 ..) . E.g. for Windows Server 2003 R2 Enterprise Edition, we get "Win NT 5.2"
  If the operating system is no Windows NT system the function returns the error value "No OS of Windows NT type"

- **GetMsVersionInfo [W]**
  returns for Windows systems the internal version info as indicated by the API, e.g. a Windows 7 system produces the result: “6.1”. For Windows 11 we get “10.0”
  so see also: **GetMsVersionName**

#### Table 3. Windows Versions

<table>
<thead>
<tr>
<th>GetMsVersionInfo</th>
<th>Windows Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>Windows 2000</td>
</tr>
<tr>
<td>5.1</td>
<td>Windows XP (Home, Prof)</td>
</tr>
<tr>
<td>5.2</td>
<td>XP 64 Bit, 2003, Home Server, 2003 R2</td>
</tr>
<tr>
<td>6.0</td>
<td>Vista, 2008</td>
</tr>
<tr>
<td>6.1</td>
<td>Windows 7, 2008 R2</td>
</tr>
<tr>
<td>6.2</td>
<td>Windows 8, 2012</td>
</tr>
<tr>
<td>6.3</td>
<td>Windows 8.1, 2012 R2</td>
</tr>
<tr>
<td>10.0</td>
<td>Windows 10, Windows 11, 2016, 2019, 2022</td>
</tr>
</tbody>
</table>

see also **GetMSVersionMap**
• *GetMsVersionName* [W] // since 4.12.4.35
  returns for Windows systems the *marketing* version info, e.g. a Windows 7 system produces the result: ”7.0”. Window 11 gives ”11.0”.

**Table 4. Windows Versionen**

<table>
<thead>
<tr>
<th>GetMsVersionName</th>
<th>Windows Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>Windows 2000</td>
</tr>
<tr>
<td>5.1</td>
<td>Windows XP (Home, Prof)</td>
</tr>
<tr>
<td>5.2</td>
<td>XP 64 Bit, 2003, Home Server, 2003 R2</td>
</tr>
<tr>
<td>6.0</td>
<td>Vista, 2008</td>
</tr>
<tr>
<td>7.0</td>
<td>Windows 7, 2008 R2</td>
</tr>
<tr>
<td>8.0</td>
<td>Windows 8, 2012</td>
</tr>
<tr>
<td>8.1</td>
<td>Windows 8.1, 2012 R2</td>
</tr>
<tr>
<td>10.0</td>
<td>Windows 10, 2016, 2019</td>
</tr>
<tr>
<td>11.0</td>
<td>Windows 11, 2022</td>
</tr>
</tbody>
</table>

see also *GetMSVersionMap*
ssee also *GetMsVersionInfo*

*getLinuxDistroType* : string [L]
returns the type of the running Linux distribution an can be used to determine which general syntax we have to use. It may return one of the following values:
- *debian* (Debian / Ubuntu ¬ use apt-get)
- *redhat* (RedHat / CentOs ¬ use yum)
- *suse* (~ use zypper) (see *getLinuxVersionMap*) [L]

*getMacosVersionInfo* : string //macOS Version Information //since 4.12.1.0 [M]
(see *getMacosVersionMap*) [M]

*GetSystemType* : string [W/L/M]
checks the installed OS if it can be assumed that the system is 64 Bit. In this case the value is *64 Bit System* otherwise *x86 System*.

*getOSArchitecture* // OS Architecture //since 4.12.4.17 checks the installed OS for the processor architecture it is build for. Possible values are:
- *x86_32* (Intel / AMD X86 Architecture with 32 Bit)
- *x86_64* (Intel / AMD X86 Architecture with 64 Bit)
- *arm_64* (ARM Architecture with 64 Bit e.g Apple M1)

**9.3.6. String Functions for Retrieving Environment or Command Line Data [W/L/M]**

*EnvVar (<environment variable>) : string* [W/L/M]
The function reads and returns the momentary value of a system environment variable. E.g., we can retrieve which user is logged in by `EnvVar ("Username")`.

- **ParamStr** [W/L/M]
  The function passes the the parameter string of the `opsi-script` command line i.e. the command line parameter which is indicated by `/parameter`. If there is no such parameter ParamStr returns the empty string.

- **getLastExitCode : string (exitcode)** [W/L/M]
  Returns a string that contains the value of the exitcode of the last process called by a WinBatch / DosBatch / ExecWith section. When using a DosBatch or ExecWith section, you will normally get the exitcode from the interpreter that was called. To get the exitcode of your script, you have to define it explicitly.

Example:

```plaintext
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]
  Returns the SID for a given user (possibly with domain prefix in the form DOMAIN\USER).

- **GetUsercontext** [W]
  Returns the string which was given to the `opsi-script` by the optional parameter `/usercontext`. If this parameter was not used, the returned string is empty.

### 9.3.7. Reading Values from the Windows Registry and Transforming Values into Registry Format [W]

- **getRegistryValue (keystr, varstr) : string** [since 4.12.0.16] [W]
  Tries to use `keystr` as Registry key and open it and read there the variable `varstr` and return the value of this variable as a string. If there is no registry key `keystr` or the variable `varstr` does not exist the function produces a warning message in the log file and returns the empty string. If `varstr` is an empty string, the default entry of the key will be returned. By Default the registry access mode is `sysnative`. Using the optional third parameter `access str`, the access mode can be explicitly given. In this case it has to be one of the following values: 32bit, sysnative, 64bit. (see also: Chapter 64 Bit)

Example:

```plaintext
getRegistryValue("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon", "Shell")
getRegistryValue("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon", "Shell","64bit")
```
GetRegistryStringValue("[key] var") : string

This command is discouraged, please use: getRegistryValue

tries to interpret the passed String value as an expression of format

\( [KEY] X \)

Then, the function tries to open the registry key \( KEY \), and, in case it succeeds, to read and return the String value that belongs to the registry variable name \( X \).

E.g.

```
GetRegistryStringValue (*[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon] Shell*)
```

usually yields "Explorer.exe", the default Windows shell program.

If there is no registry key \( KEY \) or the variable \( X \) does not exist the function produces a warning message in the log file and returns the empty string.

For example: If we made a standard entry with the value standard entry at the key HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-test\test-4.0, we will get with

```
Set $CompValue$ = GetRegistryStringValue32 (*[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-script-test\test-4.0]*)
```

the following log:

```
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 ([key] var) : string

¬ see Chapter 64 Bit  
¬ see also : getRegistryValue
```

```
GetRegistryStringValue64 ([key] var) : string

¬ see Chapter 64 Bit  
¬ see also : getRegistryValue
```

```
GetRegistryStringValueSysNative ([key] var) : string

¬ see Chapter 64 Bit  
¬ see also : getRegistryValue
```

```
RegString(<string>)

is useful for transforming path names into the format which is used in the Windows registry. That is, any backslash is duplicated. E. g.,

```
RegString ("c:\windows\system\"")
```

yields

"c:\windows\system\"
9.3.8. Reading Values from ini files [W/L/M]

For historical reasons, there are three functions for reading values from configuration files which have ini file format. Since opsi 3.0 the specific product properties are retrieved from the opsi configuration demon (that may fetch it from a configuration file or from any other backend data container).

In detail:
Ini file format means that the file is a text file and is composed of "sections" each containing key value pairs:

```
[section1]
Varname1=Value1
Varname2=Value2
...
[section2]
...
```

The most general function reads the value belonging to some key in some section of some ini file. Any parameter can be given as an arbitrary String expression:

* `GetValueFromInifile (file, section, key, default value) : string` [W/L/M]
  The function tries to open the ini file `FILE`, retrieve the requested `SECTION` and find the value belonging to the specified `KEY` which the function will return. If any of these operations fail `DEFAULTVALUE` is returned.

The second function borrows its syntax from the ini file format itself, and may sometimes be easier to use. But since this syntax turns complicated in more general circumstances it is discouraged. The syntax reads:

* `GetIni ( <Stringausdruck> [ <character sequence> ] <character sequence> )` (discouraged, use `GetValueFromInifile`) The `<String expression>` is interpreted as file name, the first `<character sequence>` as section name, the second as key name.

9.3.9. Reading Product Properties [W/L/M]

* `GetProductProperty ( <PropertyName>, <DefaultValue> )`
  where `$PropertyName$` and `$DefaultValue$` are String expressions. If `opsi-script` is connected to the opsi configuration service the product property is retrieved from the service.
  If there is no connection to the opsi server, then the resulting string is constructed in the following way:
  Since 4.12.4.32 it is checked if there ist a file `properties.conf` in the `%ScriptPath%` directory. If this file is found, we try to get the value from this file. The file will be interpreted as list of key=value pairs. In case of a string the entry should have the pattern `<property name>=<string value>` for example: `myproperty=myentry`.
  If there is no file `properties.conf` or it does not contain the searched entry, so value `<default value>` is returned.

The product properties can be used to configure variants of an installation.

E.g. the opsi UltraVNC network viewer installation may be configured using the options

* `viewer = <yes> | <no>`
  * `policy = <factory_default> |`

The installation script branches according to the chosen values for these options which can be retrieved by

```
GetProductProperty("viewer", "yes")
GetProductProperty("policy", "factory_default")
```
Since 4.12.4.32: if the file `%ScriptPath%\properties.conf` exists with the content:

```bash
propstr = from file
proplist = ['from file', 'huhu']
```

the following script will be (with out opsi service) successful:

```bash
[Actions]
DefStringList $list$
DefVar $str$
set $str$ = GetProductProperty('propstr','')
if $str$ = "from file"
  comment "got it"
else
  comment "failed"
endif
set $list$ = GetProductPropertyList('proplist','')
if takeString(0,$list$) = "from file"
  comment "got it"
else
  comment "failed"
endif
```

- **GetConfidentialProductProperty** ( `<PropertyName>`, `<DefaultValue>` ) //since 4.11.5.2
  like **GetProductProperty** but handles the resulting value as confidential string.
  Useful for getting passwords without logging. see also **SetConfidential**
  see also: **asConfidential** (string)
  see also: **asConfidential** (list)

- **IniVar** (<PropertyName>)
  (discouraged: use GetProductProperty)

### 9.3.10. Retrieving Data from etc/hosts [W/L/M]

- **GetHostName** (<string>)
  returns the host name to a given IP address as it is declared in the local hosts file. If the operating system is "Windows_NT" (according to environment variable OS) "%systemroot%\system32\drivers\etc" is assumed as host file location, otherwise "C:\Windows\".

- **GetHostsAddr** (<string>)
  tells the IP address to a given host or alias name.

### 9.3.11. String Handling [W/L/M]

- **ExtractFilePath** (<path>): string
  interprets the passed String value as file or path name and returns the path part (the string up to the last path delimiter, including it).

Examples:
set $ConstTest$ = "C:\program files\test\"
Set $tmp$ = "C:\program files\test\test.exe"
set $CompValue$ = ExtractFilePath($tmp$)
if ($ConstTest$ = $CompValue$)
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif

* ExtractFileExtension (<path>) : string // since 4.12.1 [W/L/M]
interprets the passed String value as file or path name and returns the extension part (the string after the last dot (.), including it).

Examples:

set $ConstTest$ = ".exe"
Set $tmp$ = "C:\program files\test\test.exe"
set $CompValue$ = ExtractFileExtension($tmp$)
if ($ConstTest$ = $CompValue$)
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif

* ExtractFileName (<path>) : string // since 4.12.1 [W/L/M]
interprets the passed String value as file or path name and returns the file name part (the string after the last path delimiter, not including it).

Examples:

set $ConstTest$ = "test.exe"
Set $tmp$ = "C:\program files\test\test.exe"
set $CompValue$ = ExtractFileName($tmp$)
if ($ConstTest$ = $CompValue$)
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif

* forcePathDelims (<path string>) : <path string> // since 4.12.4.21 [W/L/M]
interprets the passed String value as path name and replaces all path delimiter to the char that is specific for the running Operating System (Windows: \\, Linux und macOS: /).

* resolveSymlink (<file name>) : <file name> // since 4.12.4.21 [W/L/M]
If the given file <file name> is a symbolic link its target will be (recursiv) resolved and the result will be returned. In any other case the return value ist the given <file name>.

* [FileExists]
* [FileOrFolderExists]
* [DirectoryExists]
* [fileIsSymlink]

* StringSplit ("STRINGWERT1, STRINGWERT2, INDEX")
deprecated: use splitString/takeString
takeString (index, list) : string

returns from a string list list the string with the index index.
Often used in combination with splitString: takeString(index, splitString(string1, string2))
(see also String List Functions and String List Processing).
The result is produced by slicing string1 where each slice is delimited by an occurrence of string2, and then taking the slice with index index (where counting starts with 0).

Example:
```
takeString(3, splitString("\server\share\directory", ")
```
returns "share", the given string split at "\" returns the string list:
Index 0 - "" (empty string), because there is nothing before the first "\"
Index 1 - "" (empty string), because there is nothing before the second "\"
Index 2 - "server"
Index 3 - "share"
Index 4 - "directory"
takeString counts downward, if the index is negative, starting with the number of elements. Therefore,
```
takeString(-1, $list1$
```
denotes the last element of String list $list1$.
see also : setStringInListAtIndex

SubstringBefore(string1, string2)
(yielded: use splitString/takeString) yields the sequence of characters of stringValue1 up to the beginning of stringValue2.
Example:
```
SubstringBefore ("C:\programme\staroffice\program\soffice.exe", "\program\soffice.exe")
```
returns "C:\programme\staroffice".

getIndexFromListByContaining(list : stringlist, search string : string) : number
(since 4.12.0.13 [W/L/M]
Returns a string that holds the index of the first string in list which contains search string.
Retruns a empty string if no matching string is found.
The check is performed case-insensitive.
see also : takeFirstStringContaining
takeFirstStringContaining(list, search string) : string
returns the first string from list which contains search string.
Retruns a empty string if no matching string is found.
see also : getIndexFromListByContaining

trim(string) : string
cuts leading and trailing white space from string.

lower(string) : string
returns <string> with lower case.

• **upper(<string>)** [W/L/M]
  returns <string> with upper case.

• **contains(<str>, <substr>) : bool** //since 4.11.3: true if <substr> in <str> [W/L/M]
  A boolean function which returns true if <str> contains <substr>. This function is case sensitive.
  Available since 4.11.3
  Example:

```opsi
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>) : string** //since 4.11.3 [W/L/M]
  returns a string, which has all occurrences of <oldPattern> replaced with <newPattern> given then input string <string>. The pattern match is case insensitive.
  Example:

```opsi
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>) : string (number)** //since 4.11.3 [W/L/M]
  Returns the number of chars in in <string>
  Example:

```opsi
set $tmp$ = "123456789"
set $ConstTest$ = "$9"
set $CompValue$ = strLength($tmp$)
if $ConstTest$ = $CompValue$
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif
set $tmp$ = ""
set $ConstTest$ = "0"
set $CompValue$ = strLength($tmp$)
if $ConstTest$ = $CompValue$
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif
```
• `strPos(<string>, <sub string>) : string //since 4.11.3 [W/L/M]`
returns the first position of `<sub string>` in `<string>`. If `<sub string>` is not found, then "0" is the return value. The function is case sensitive.

Example:

```plaintext
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>) : string //since 4.11.3 [W/L/M]`
returns the part of `<string>` starting with `<start pos>` and include the next `<number of chars>` chars. If there are fewer than `<number of chars>` after `<start pos>`, then the returned string will be the rest of the chars after `<start pos>`. The counting of chars starts with 1.

Example:

```plaintext
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>) : string //since 4.11.2.1 [W/L/M]`
returns the unquoted version of `<string>`, if `<string>` is quoted with `<quote-string>`. Only one char (the first char) of `<quote-string>` is accepted as a quote char. The leading white spaces are ignored.
see also: [unquote2]
set $ConstTest$ = "b"
set $CompValue$ = unquote("'b'", "'")
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($CompValue$,"]")
set $CompValue$ = unquote(unquote($CompValue$,"["),"]")
set $CompValue$ = unquote($CompValue$,"[")
set $CompValue$ = unquote($CompValue$,"]")
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

• unquote(<string>,<quote-string>) : string [since 4.11.5.2 [W/L/M]]
  Acts like unquote(<string>,<quote-string>) with the following differences:
  If <quote-string> contains only one char, so this char will be used as start quote char and end quote char. If <quote-string> contains two chars, so the first char will be used as start quote char and the second char as end quote char. Example: a <quote-string> like "()" will unquote a string like (hello).
  The function returns the unchanged <string> if not (start quote char AND end quote char) is found. see also : [unquote]

• HexStrToDecStr (<hexstring>) : string [W/L/M]
  returns the decimal representation of the input string if this was the hexadecimal representation of an integer. Leading chars like 0x or $ will be ignored. In case of a converting error the function returns a empty string.

• DecStrToHexStr (<decstring>, <hexlength>) : string [W/L/M]
  returns a <hexlength> long string with the the hexadecimal representation of <decstring> if this was the decimal representation of an integer. In case of a converting error the function returns a empty string.
base64EncodeStr(<string>) : string [W/L/M]
returns the base64 encoded value of <string>.

base64DecodeStr(<string>) : string [W/L/M]
returns the base64 decoded value of <string>.

encryptStringBlow(<keystring>,<datastring>) : string [W/L/M]
Encrypted <datastring> with the Key <keystring> under application of Blowfish and returns the encrypted value.

decryptStringBlow(<keystring>,<datastring>) : string [W/L/M]
Decrypts <datastring> with the Key <keystring> under the application of Blowfish and returns the decrypted value.
Chapter 9. Syntax and Meaning of Primary Sections of an opsi-script Script [W/L/M] 88 / 281

```
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 [W/L/M]
  Returns the md5sum that under path to file was found.
  In case of error returns an empty String.

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/M]
  assumes that str is encoded in from and returns the in to encoded version of str.
  from and to are encodings as listet in chapter opsi-script encoding.
  see also: [reencodestrlist] siehe auch: [loadTextFile]
  siehe auch: [strLoadTextFileWithEncoding]
  siehe auch: [loadUnicodeTextFile]
  siehe auch: [loadTextFileWithEncoding]
  see also: Encoding related functions

• strLoadTextFile (filename) : string [W/L/M]
  returns the first line of filename as String.
  see also: [strLoadTextFileWithEncoding]

• strLoadTextFileWithEncoding (filename, encoding) : string [W/L/M]
  returns the first line of filename as String reencodes from encoding.
  see also: [loadTextFile]
  see also: [strLoadTextFile]
  see also: [loadUnicodeTextFile]
  see also: [loadTextFileWithEncoding]
  see also: opsi-script encoding

• GetShortWinPathName(longpath string) //since 4.11.5.2 [W]
  Returns the short path (8.3) from longpath string. If there is no short path for longpath string, so you will get an empty string.
  Example: GetShortWinPathName("C:\Program Files (x86)") returns "C:\PROGRA~2"

9.3.12. Other String Functions

• RandomStr (usespecialchars): string [W/L/M]
```
returns a random String of length 10 where upper case letters, lower case letters and digits are mixed (for creating passwords). More exactly: if usespecialchars = true, it will create a string of 2 lower case chars, 2 upper case chars, 2 special chars and 4 digits. The possible special chars are:

!,$,(,*,+/,;=,?[,]\{\},ß,~,§,°

if usespecialchars = false, it will create a string of 3 lower case chars, 3 upper case chars and 4 digits.

• RandomStrWithParameters (<minLength>,<nLowerCases>,<nUpperCases>,<nDigits>,<nSpecialChars>): string [W/L/M]
returns a random String (useful for creating passwords) according to the input configuration in the parameters, where:

- <minLength>: the length of the string,
- <nLowerCases>: the count of lower case letters wanted,
- <nUpperCases>: the count of upper case letters wanted,
- <nDigits>: the count of digits wanted,
- <nSpecialChars>: the count of special characters wanted.
Possible special chars are: !,$,(,*,+/,;=,?[,]\{\},ß,~,§,°

• RandomIntStr(<number str>) : string [W/L/M]
returns a number between 0 and <number str> as string.

• CompareDotSeparatedNumbers(<string1>, <string2>) : string [W/L/M]
compares two strings of the form <number>[.<number>[.<number>[.<number>]]]
It returns "0" if the strings are equal, "1" if <string1> is higher and "-1" if <string1> is lower than <string2>.
see also: CompareDotSeparatedNumbers(<str1>,<relation str>,<str2>) : [CompareDotSeparatedNumbers bool]
see also: [CompareDotSeparatedStrings str]

Example:
The Code:
leads to the following log:
**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
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
Then
  comment: passed
  comment: 1.2.31.4.5 is higher than 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
Then
  comment: passed
  comment: 1.2.3.4.5 is lower than 1.2.13.4.5
Else
EndIf

• **CompareDotSeparatedStrings**(string1, string2) : string [W/L/M]
  compares two strings of the form string.<string>[.<string>[.<string>]]
  It returns "0" if the strings are equal, "1" if string1 is higher and "-1" if string1 is lower than string2. The function is not case sensitive.
  see also : [CompareDotSeparatedStrings_bool]
  see also : [CompareDotSeparatedNumbers_bool]

Example:
The Code:
leads to the following 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"
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 "$string1" is higher than "$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 "$string1" is lower than "$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 "$string1" is lower than "$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 "$string1" is higher than "$string2"
  comment "using CompareDotSeparatedStrings give wrong results on numbers"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif
```

Chapter 9. Syntax and Meaning of Primary Sections of a \textit{opsi-script} Script [W/L/M]
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 than 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 than 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
$\text{ConstTest} = \text{CompValue}$   <<< result true
($\text{ConstTest} = \text{CompValue}$)   <<< result true
Then
    comment: passed
    comment: 1.2.13.4.5 is lower than 1.2.3.4.5
    comment: using CompareDotSeparatedStrings give wrong results on numbers
Else
EndIf

Set  $\text{string1}$ = "1.2.3.4.5"
The value of the variable "$\text{string1}$" is now: "1.2.3.4.5"

Set  $\text{string2}$ = "1.2.13.4.5"
The value of the variable "$\text{string2}$" is now: "1.2.13.4.5"

Set  $\text{ConstTest} = \text{1}$
The value of the variable "$\text{ConstTest}$" is now: "1"

Set  $\text{CompValue} = \text{CompareDotSeparatedStrings}(\text{string1}, \text{string2})$
The value of the variable "$\text{CompValue}$" is now: "1"

If
$\text{ConstTest} = \text{CompValue}$   <<< result true
($\text{ConstTest} = \text{CompValue}$)   <<< result true
Then
    comment: passed
    comment: 1.2.3.4.5 is higher than 1.2.13.4.5
    comment: using CompareDotSeparatedStrings give wrong results on numbers
Else
EndIf

- **getDiffTimeSec** [W/L/M]
  returns a string with the integer number of seconds since the last call of **marktime**.
  Available since 4.11.3.1

- **timeStampAsFloatStr** : string (Floating Number - format: *days.decimal days*) //since 4.11.6 [W/L/M]
  Gives Date and Time from now as string that contains a decimal number in the format: *days.decimal days*. This Format make it easier to calculate time differences.

- **SidToName(<well known sid>)** [W]
  returns a string with the localized name of the group with the <well known sid>. For example, if <well known sid> is equal to S-1-5-32-544 then **SidToName** returns Administrators.
  Available since 4.11.3.1

- **GetMyIpByTarget(<target ip addr>)** [W/L/M]
  returns a list of interface IP-addresses, which are trying to reach the operating system at <target ip addr>. This function returns a value that is safer than the constant %IPAddress%.
  Since Version 4.11.3.1
  Example:
  ```
  set $\text{CompValue} = \text{GetMyIpByTarget("opsiServer")}
  ```
  see also: [GetIpByName]
  see also: [IPAddress]

- **GetIpByName(<ip addr / ip name>)** [W/L/M]
  returns the IP-addresses of the computers with the <ip addr / ip name>
  Since 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

see also : [GetMyIpByTarget]

**stringinput(< message str>,< boolstr confidential>) : string //since 4.12.1.2 [W/L/M]**

Interactive function.
Interactive function to get a string from the user. It outputs the < message str> and waits for the Input and returns the input string.
In the grafical mode this is done in a modal Window, in the non grafical mode at the command line.
If < boolstr confidential> = "true" then the input is masked by "*". A button with a eye icon can be used to get a unmasked, readable display.
If < boolstr confidential> = "false" the the input is readable.

**replaceOpsiConstants(<string>) : string //since 4.12.3.6 [W/L/M]**

returns a string, which has all occurrences of opsi constants in <string> replaced with their value.
see also : [replaceOpsiConstants_list]

**fileHasBom (<file name>) : boolean //since 4.12.4.17 [W/L/M]**

This function reads the first 4 bytes of the input file and detects if it has a BOM or not.
If yes, the return value is true else false.
You can know more about BOM encodings here : https://en.wikipedia.org/wiki/Byte_order_mark
see also : opsi-script encoding
see also: Encoding related functions see also : [strLoadTextFileWithEncoding]
see also : [loadUnicodeTextFile]
see also : [loadTextFileWithEncoding]

9.3.13. (String-) Functions for Licence Management [W/L/M]

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

asks the opsi service via the function getAndAssignSoftwareLicenseKey for a reservation of a licence for the client.
The pool from which the licences is taken may be explicitly given by its ID or is identified via an associated product ID or Windows Software Id (possible, if these associations are defined in the licences configuration).
*poolId*, *productId*, *windowsSoftwareId* are Strings (resp. String expressions).
If no *poolId* is explicitly given, the first parameter has to be an empty String "". The same procedure is done with other not explicit given Ids.
The function returns the licence key that is taken from the pool.

Examples:
set $mykey$ = DemandLicenseKey("pool_office2007")
set $mykey$ = DemandLicenseKey("", "office2007")
set $mykey$ = DemandLicenseKey("", "", "{3248F0A8-6813-11D6-A77B}")

• `FreeLicense('poolId [, productId [,windowsSoftwareId]])`' asks the opsi service via the function freeSoftwareLicense to release the current licence reservation. The syntax is analogous to the syntax for `DemandLicenseKey`.

Example:

DefVar $opsiresult$
set $opsiresult$ = FreeLicense("pool_office2007")

$opsiresult$ becomes the empty String, if no error occurred, and, if an error occurred, the error info text.

9.3.14. Retrieving Error Infos from Service Calls [W/L/M]

• `getLastServiceErrorClass` returns, as its name says, the class name of the error information of the last service call. If the last service call did not produce an error the function returns the value "None".

• `getLastServiceErrorMessage` returns the message String of the last error information resp. "None". Since the message String is more likely to be changed, it is recommended to base script logic on the class name.

Example:

if getLastServiceErrorClass = "None"
  comment "kein Fehler aufgetreten"
endif

9.4. String List Functions and String List Processing [W/L/M]

A String list (or a String list value) is a sequence of String values. For this kind of values we have the variable of type String list. Since 4.12.4.32 it is also possible to pass an optional initial value. They are defined by the statement

DefStringList <VarName> [= <initial value>]

A String list value may be assigned to String list variable:

Set <VarName> = <StringListValue>

String list values can be given only as results of string list expressions. These string list expressions can be constructed on different ways:

• by a function that returns a string list
• by a string list variable
• since 4.12.4.32 by a json style string that can be read as string list

For the following examples we declare a String list variable $list1$:
Since 4.12.4.32 the simplest way is a string in json style that can be read as string list:
\["<string>\]  
Example:
```bash
set $list1$ = '\"[\"ab\",\"cd\",\"de\""]'
```

The same result we get by using the function `CreateStringList`:
```bash
set $list1$ = createstringlist\("ab\",\"cd\",\"de\")
```

There are many ways to create or capture String lists, and many options for processing them, often yielding new String lists. They are presented in the following subsections.

If we refer to variables named like String0, StringVal, .. it is meant that these represent any String expressions.

We start with a special and rather useful kind of String lists: maps – also called hashes or associative arrays – which consist of a lines of the form \texttt{KEY}=\texttt{VALUE}. In fact, each map should establish a function which associates a \texttt{VALUE} to a \texttt{KEY}, and any \texttt{KEY} should occur at most once as the first part of a line (whereas different \texttt{KEY}'s may be associated with identical \texttt{VALUE} parts).

### 9.4.1. Info Maps

- **getHWBiosInfoMap** //since 4.11.4 [L/W]
  - get hardware information from BIOS and writes them to hash map string list.
  - There are the folowing keys: (example):

  ```
  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.USDID=303033343144323730434336FPPPPPPPFP
  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
  ```

- **getMacosVersionMap** : stringlist //macOS Version map //since 4.12.1.0 [M]

Example:
```bash
set $macosinfomap$ = getMacosVersionMap
```

gives (for example) the log:
The value of the variable "$macOSinfomap$" is now:

(string 0)Release=11.0
(string 1)Build=20A5364e
(string 2)kernel name=Darwin
(string 3)node name=vmmac1100onmm1.uib.local
(string 4)kernel release=20.1.0
(string 5)kernel version=Darwin Kernel Version 20.1.0: Fri Aug 28 20:45:30 PDT 2020; root:xnu-7195.40.65.0.2~61/RELEASE_X86_64
(string 6)machine=x86_64
(string 7)processor=i386
(string 8)operating system=macOS

• **getLinuxVersionMap : stringlist //since 4.11.4 [L]**

get OS information and writes them to hash map string list.
There are the following keys: (example):

Distributor ID=Ubuntu
Description=Ubuntu 12.04.2 LTS
Release=12.04
Codename=precise
kernel name=Linux
node name=detlefvm05
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

• **GetMSVersionMap : stringlist [W]**

get OS information and writes them to hash map string list.

see also [GetMsVersionInfo](#)
see also [GetMsVersionName](#)

There are the following keys: * 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

The Results from **suite_mask** and **product_type_nr** are integers that can be build by * or operations of the following values.

**product_type_nr**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000001</td>
<td>VER_NT_WORKSTATION</td>
</tr>
<tr>
<td>0x0000002</td>
<td>VER_NT_DOMAIN_CONTROLLER</td>
</tr>
<tr>
<td>0x0000003</td>
<td>VER_NT_SERVER</td>
</tr>
</tbody>
</table>

**SuiteMask**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001</td>
<td>VER_SUITE_SMALLBUSINESS</td>
</tr>
<tr>
<td>0x00000002</td>
<td>VER_SUITE_ENTERPRISE</td>
</tr>
<tr>
<td>0x00000004</td>
<td>VER_SUITE_BACKOFFICE</td>
</tr>
<tr>
<td>0x00000008</td>
<td>VER_SUITE_COMMUNICATIONS</td>
</tr>
<tr>
<td>0x00000010</td>
<td>VER_SUITE_TERMINAL</td>
</tr>
<tr>
<td>0x00000020</td>
<td>VER_SUITE_SMALLBUSINESS_RESTRICTED</td>
</tr>
<tr>
<td>0x00000040</td>
<td>VER_SUITE_EMBEDDED</td>
</tr>
<tr>
<td>0x00000080</td>
<td>VER_SUITE_DATACENTER</td>
</tr>
<tr>
<td>0x00000100</td>
<td>VER_SUITE_SINGLEUSERS</td>
</tr>
<tr>
<td>0x00000200</td>
<td>VER_SUITE_PERSON</td>
</tr>
<tr>
<td>0x00000400</td>
<td>VER_SUITE_SERVERAPPLIANCE</td>
</tr>
</tbody>
</table>

• **ReleaseID** which gives you the sub release of **Windows 10** like e.g. **1511**. The Value comes from the Registry:
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ReleaseID

- **prodInfoText** which gives you a string to the edition type like e.g. `PRODUCT_PROFESSIONAL`.
- **prodInfoNumber** which gives you a string with a decimal number of the edition type like e.g. `48`.

### ProdInfoNumber und ProdInfoText

<table>
<thead>
<tr>
<th>DecNum</th>
<th>HexNum</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>An unknown product</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>Ultimate Edition*</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>Home Basic Edition</td>
</tr>
<tr>
<td>03</td>
<td>03</td>
<td>Home Premium Edition</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>Enterprise Edition</td>
</tr>
<tr>
<td>05</td>
<td>05</td>
<td>Home Basic Edition</td>
</tr>
<tr>
<td>06</td>
<td>06</td>
<td>Business Edition</td>
</tr>
<tr>
<td>07</td>
<td>07</td>
<td>Server Standard Edition (full installation)</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td>Server Datacenter Edition (full installation)</td>
</tr>
<tr>
<td>09</td>
<td>09</td>
<td>Small Business Server</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>Server Enterprise Edition (full installation)</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>Starter Edition</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>Server Datacenter Edition (core installation)</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>Server Standard Edition (core installation)</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Server Enterprise Edition (core installation)</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Server Enterprise Edition for Itanium-based Systems</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Business Edition</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Web Server Edition (full installation)</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Cluster Server Edition</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Home Server Edition</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Storage Server Express Edition</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Storage Server Standard Edition</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Storage Server Workgroup Edition</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>Storage Server Enterprise Edition</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Server for Small Business Edition</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>Small Business Server Premium Edition</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>PRODUCT_HOME_PREMIUM_N</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>PRODUCT_ENTERPRISE_N</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>PRODUCT ULTIMATE_N</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>PRODUCT_WEB_SERVER_CORE</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>Windows Essential Business Server Management Server</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>Windows Essential Business Server Security Server</td>
</tr>
<tr>
<td>Line</td>
<td>Column</td>
<td>Product Name</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>Windows Essential Business Server Messaging Server</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>Server Foundation</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>PRODUCT_HOME_PREMIUM_SERVER</td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>PRODUCT_SERVER_FOR_SMALLBUSINESS_V</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>Server Standard Edition without Hyper-V (full installation)</td>
</tr>
<tr>
<td>37</td>
<td>25</td>
<td>Server Datacenter Edition without Hyper-V (full installation)</td>
</tr>
<tr>
<td>38</td>
<td>26</td>
<td>Server Enterprise Edition without Hyper-V (full installation)</td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>Server Datacenter Edition without Hyper-V (core installation)</td>
</tr>
<tr>
<td>40</td>
<td>28</td>
<td>Server Standard Edition without Hyper-V (core installation)</td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>Server Enterprise Edition without Hyper-V (core installation)</td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>PRODUCT_PROFESSIONAL</td>
</tr>
<tr>
<td>49</td>
<td>31</td>
<td>PRODUCT_PROFESSIONAL_N</td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>PRODUCT_SB_SOLUTION_SERVER</td>
</tr>
<tr>
<td>51</td>
<td>33</td>
<td>PRODUCT_SERVER_FOR_SB_SOLUTIONS</td>
</tr>
<tr>
<td>52</td>
<td>34</td>
<td>PRODUCT_STANDARD_SERVER_SOLUTIONS</td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td>PRODUCT_STANDARD_SERVER_SOLUTIONS_CORE</td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>PRODUCT_SB_SOLUTION_SERVER_EM</td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td>PRODUCT_SERVER_FOR_SB_SOLUTIONS_EM</td>
</tr>
<tr>
<td>56</td>
<td>38</td>
<td>PRODUCT_SOLUTION_EMBEDDEDSERVER</td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td>PRODUCT_SOLUTION_EMBEDDEDSERVER_CORE</td>
</tr>
<tr>
<td>59</td>
<td>3B</td>
<td>PRODUCT_ESSENTIALBUSINESS_SERVER_MGMT</td>
</tr>
<tr>
<td>60</td>
<td>3C</td>
<td>PRODUCT_ESSENTIALBUSINESS_SERVER_ADDL</td>
</tr>
<tr>
<td>61</td>
<td>3D</td>
<td>PRODUCT_ESSENTIALBUSINESS_SERVER_MGMTSVCC</td>
</tr>
<tr>
<td>62</td>
<td>3E</td>
<td>PRODUCT_ESSENTIALBUSINESS_SERVER_ADDLSVC</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>PRODUCT_SMALLBUSINESS_SERVER_PREMIUM_CORE</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>PRODUCT_CLUSTER_SERVER_V</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>PRODUCT_EMBEDDED</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>PRODUCT_STARTER_E</td>
</tr>
<tr>
<td>67</td>
<td>43</td>
<td>PRODUCT_HOME_BASIC_E</td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>PRODUCT_HOME_PREMIUM_E</td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td>PRODUCT_PROFESSIONAL_E</td>
</tr>
<tr>
<td>70</td>
<td>46</td>
<td>PRODUCT_ENTERPRISE_E</td>
</tr>
<tr>
<td>71</td>
<td>47</td>
<td>PRODUCT_ULTIMATE_E</td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>PRODUCT_ENTERPRISE_EVALUATION</td>
</tr>
<tr>
<td>84</td>
<td>54</td>
<td>PRODUCT_ENTERPRISE_N_EVALUATION</td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>PRODUCT_CORE_N</td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>PRODUCT_CORE_COUNTRYSPECIFIC</td>
</tr>
</tbody>
</table>
Example:
The Code

```plaintext
DefStringList $INST_Resultlist$
DefStringList $INST_Resultlist2$
message "getMSVersionMap"
comment "get value by winst function"
set $INST_Resultlist$ = getMSVersionMap
```

produces the following log:

```plaintext
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
```

Background infos for getMSVersionMap


- `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]`

retrieves the version infos built into the file FILENAME and writes it to a Stringlist map.

At this moment, there exist the keys,

- 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

Usage: If we define and call

\[
\text{DefStringList FileInfo} \\
\text{DefVar $InterestingFile$} \\
\text{Set $InterestingFile$} = \text{"c:\program files\my program.exe"} \\
\text{set FileInfo} = \text{getFileInfoMap($InterestingFile$)}
\]

we get the value associated with key "FileVersion" from the call

\[
\text{DefVar $result$} \\
\text{set $result$} = \text{getValue("FileVersion", FileInfo)}
\]

(for the function getValue cf. Simple String Values generated from String Lists or Files).

Example:
The code:

\[
\text{set $InterestingFile$} = \"%OpsiScriptDir%\winst.exe\" \\
\text{if not (FileExists($InterestingFile$))} \\
\text{set $InterestingFile$} = \"%OpsiScriptDir%\winst32.exe\" \\
\text{endif} \\
\text{set $INST_Resultlist$} = \text{getFileInfoMap($InterestingFile$)}
\]

produce the 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-script
(string 15)ProductVersion=4.0
(string 16)SpecialBuild=

* GetLocaleInfoMap [W]

retrieves the system informations on the locale and writes it to a Stringlist map.

At this moment, there exist the keys:

- language_id_2chars (two-letter version of the system default language name)
- language_id (three-letter version of it, including subtype of language 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 Language_Region (Posix Style)
- system_default_lang_region Language-Region (BCP 47 Style)

The system_default keys gives information about the language of the installed OS. The other keys give information about the locale of the GUI.
Example:
The code:

```plaintext
message *Locale Infos*
set $INST_Resultlist$ = GetLocaleInfoMap
```

produces e.g the log:

```plaintext
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
```

Usage: If we define and call

```plaintext
DefStringList $languageInfo$
set $languageInfo$ = GetLocaleInfoMap
```

we get the value associated with key "language_id_2chars" from the call

```plaintext
DefVar $result$
set $result$ = getValue("language_id_2chars", $languageInfo$)
```

(for the function getValue cf. Simple String Values generated from String Lists or Files). We may now write scripts using a construct like

```plaintext
if getValue("language_id_2chars", languageInfo) = "DE"
    ; installiere deutsche Version
else
    if getValue("language_id_2chars", languageInfo) = "EN"
        ; installiere englische Version
    endif
endif
```

Background infos for GetLocaleInfoMap:

- bcp 47 validator: [http://schneegans.de/lv/?tags=de-de-1996&format=text](http://schneegans.de/lv/?tags=de-de-1996&format=text)
- [http://www.iana.org/assignments/language-subtag-registry](http://www.iana.org/assignments/language-subtag-registry)

*getLocaleInfo*
(deprecated): use GetLocaleInfoMap.
see also: [GetLocaleInfoMap]
• **getProductMap** // since 4.11.2.4 [W/L/M]
  returns a info map of the opsi product you are just installing.
  It works only if **opsi-script** is running in opsi service mode.
  keys are: id, name, description, advice, productversion, packageversion, priority, installationstate, lastactionrequest, lastactionresult, installedversion, installedpackage, installedmodificationtime, actionrequest

Example:

```plaintext
set $INST_Resultlist$ = getProductMap
set $string1$ = getValue("id", $INST_Resultlist$)
```

produces e.g the log:

```plaintext
Set $INST_Resultlist$ = getProductMap
retrieving strings from getProductMap [switch to loglevel 7 for debugging]
(string 0)id=opsi-script-test
(string 1)name=opsi-script test
(string 2)description=Test and example script for opsi-script
(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-script test
(string 2)description=Test and example script for opsi-script
(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"
```

• **editmap(<strlist>) : stringlist // since 4.12.1.2 [W/L/M]**
  Interactive function.
  Show the given <strlist> as <key>=<value> pair list to the user and gives the possibility to change values. Returned ist the edited version of <str list>, after the user user finished editing.
  In the grafical mode this is done in a modal Window, in the non grafical mode at the command line.

getListFromWMI(<wmi namespace str>, <wmi class str>, <property list>, <condition str>) : stringlist //since 4.12.1.0 [W]
Returns an info map of <wmi class str> that is limited to the properties in <property list> and also is limited by the <condition str>.
If <property list> is empty, all properties of <wmi class str> will be returned.
Attention: If <property list> contains properties that don’t exist in the called WMI class, the query will fail.
Calling with properties is faster than without.
If <wmi namespace str> is empty, it defaults to root\cimv2.
If an error occurs, an empty list will be returned

Example:
set $list1$ = createStringList ('Model', 'Manufacturer')
set $str1$ = 'root\cimv2'
set $str2$ = 'Win32_ComputerSystem'
set $str3$ = ''
set $resultlist$ = getListFromWMI($str1$, $str2$, $list1$, $str3$)

produces e.g the log:

Set $list1$ = createStringList ('Model', 'Manufacturer')
The value of the variable "$list1$" is now:
| string | (0)Model |
| string | (1)Manufacturer |
Set $str1$ = 'root\cimv2'
The value of the variable "$str1$" is now: "root\cimv2"
Set $str2$ = 'Win32_ComputerSystem'
The value of the variable "$str2$" is now: "Win32_ComputerSystem"
Set $str3$ = ''
The value of the variable "$str3$" is now: ""
Set $resultlist$ = getListFromWMI($str1$, $str2$, $list1$, $str3$)
The value of the variable "$resultlist$" is now:
| string | (0)Model=HP Pavilion Desktop PC 570-p0xx |
| string | (1)Manufacturer=HP |

Example:

comment "Testing for os architecture"
set $ConstTest$ = GetSystemType
set $list1$ = createStringList ('systemtype')
set $str1$ = ''
set $str2$ = 'Win32_ComputerSystem'
set $str3$ = ''
set $resultlist$ = getListFromWMI($str1$, $str2$, $list1$, $str3$)

produces e.g the log:

comment: Testing for os architecture
Set $ConstTest$ = GetSystemType
The value of the variable "$ConstTest$" is now: "64 Bit System"
Set $list1$ = createStringList ('systemtype')
The value of the variable "$list1$" is now:
| string | (0)systemtype |
Set $str1$ = ''
The value of the variable "$str1$" is now: ""
Set $str2$ = 'Win32_ComputerSystem'
The value of the variable "$str2$" is now: "Win32_ComputerSystem"
Set $str3$ = ''
The value of the variable "$str3$" is now: ""
Set $resultlist$ = getListFromWMI($str1$, $str2$, $list1$, $str3$)
The value of the variable "$resultlist$" is now:
| string | (0)systemtype=x64-based PC |

Example:
Comment: Testing for freespace

; wmic LogicalDisk "%Systemdrive%" get freespace
set $list1$ = createStringList ('freespace')
set $str1$ = 'root\cimv2'
set $str2$ = 'Win32_LogicalDisk'
set $str3$ = 'where Name="%Systemdrive%"

markerrornumber
set $resultlist$ = getListFromWMI($str1$,$str2$,$list1$,$str3$)
if errorsOccuredSinceMark = 0
set $CompValue$ = getValue("freespace", $resultlist$)
set $CompValue$ = calculate($CompValue$ - '1')
if (HasMinimumSpace ("%Systemdrive%", $CompValue$))
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif
set $CompValue$ = calculate($CompValue$ + '10')
if (HasMinimumSpace ("C:", $CompValue$))
  set $TestResult$ = "not o.k."
  LogWarning "failed"
else
  comment "passed"
endif
endif

don't produce e.g the log:

Comment: Testing for freespace
Set $list1$ = createStringList ('freespace')
The value of the variable "$list1$" is now:
(string 0)freespace
Set $str1$ = 'root\cimv2'
The value of the variable "$str1$" is now: "root\cimv2"
Set $str2$ = 'Win32_LogicalDisk'
The value of the variable "$str2$" is now: "Win32_LogicalDisk"
Set $str3$ = 'where Name="C:\"
The value of the variable "$str3$" is now: "where Name="C:\"
Marked error number 1
Set $resultlist$ = getListFromWMI($str1$,$str2$,$list1$,$str3$)
The value of the variable "$resultlist$" is now:
(string 0)freespace=235092250624
If errorsOccuredSinceMark = 0   <<< result true
Then
Set $CompValue$ = getValue("freespace", $resultlist$)
The value of the variable "$CompValue$" is now: "235092250624"
Set $CompValue$ = calculate($CompValue$ - '1')
The value of the variable "$CompValue$" is now: "235092250623"
If Free on Disk C:\ 235.092.250.624 bytes This is more than the required amount of 235.092.250.623 bytes
HasMinimumSpace ("C:\", $CompValue$)   <<< result true
(HasMinimumSpace ("C:\", $CompValue$))   <<< result true
Then
  comment: passed
Else
EndIf
Set $CompValue$ = calculate($CompValue$ - '10')
The value of the variable "$CompValue$" is now: "235092250633"
If Free on Disk C:\ 235.092.250.624 bytes This is less than the required amount of 235.092.250.633 bytes
HasMinimumSpace ("C:\", $CompValue$)   <<< result false
(HasMinimumSpace ("C:\", $CompValue$))   <<< result false
Then
Else
  comment: passed
EndIf
EndIf

Example:
produces e.g the log:

comment: Testing for drive count
Set $list1$ = createStringList ('Name')
The value of the variable "$list1$" is now: [string 0]Name
Set $str1$ = ''
The value of the variable "$str1$" is now: ''
Set $str2$ = 'Win32_LogicalDisk'
The value of the variable "$str2$" is now: "Win32_LogicalDisk"
Set $str3$ = ''
The value of the variable "$str3$" is now: ''
Set $resultlist$ = getListFromWMI($str1$,$str2$,$list1$,$str3$)
The value of the variable "$resultlist$" is now: (string 0)Name=C:
(string 1)Name=D:
(string 2)Name=P:
Set $CompValue$ = count($resultlist$)
The value of the variable "$CompValue$" is now: "3"
Set $resultlist$ = powershellCall('get-psdrive -psprovider filesystem | select-object -expand Name')
ShellCall Executing: "C:\Windows\cmd64.exe" /C "powershell.exe get-executionpolicy"
ExitCode 0
ShellCall Executing: "C:\Windows\cmd64.exe" /C "powershell.exe set-executionpolicy RemoteSigned"
ExitCode 0
Execution of tmp-internal powershell.exe winst /sysnative
trap { write-output $_ ; exit 1 }
get-psdrive -psprovider filesystem | select-object -expand Name
exit $LASTEXITCODE
ExitCode 0
The file: c:\opsi.org\tmp\_opsiscript_Lw32Rh40.ps1 has been deleted
ShellCall Executing: "C:\Windows\cmd64.exe" /C "powershell.exe set-executionpolicy Restricted"
ExitCode 0
The value of the variable "$resultlist$" is now: [string 0]C
[string 1]D
[string 2]P
Set $ConstTest$ = count($resultlist$)
The value of the variable "$ConstTest$" is now: "3"
If $ConstTest$ = $CompValue$ <<< result true
|ConstTest$ = $CompValue$ <<< result true
Then
comment: getListFromWMI passed
Else
EndIf

• opsi-wmi-test.exe You can test your WMI query with a small helper application (opsi-wmi-test.exe). You can download it here: https://download.uib.de/opsi4.1/misc/helper/opsi-wmi-test.exe

The application works similar as the function getListFromWMI.

The user interface of opsi-wmi-test is subdivided in two sections.

At the upper section (Connection to WMI service) you are asked to input the data which is necessary to connect to the WMI service. Input here the name of the computer (Computer) addressed, the WMI namespace (NameSpace) of the WMI class which shall be used, and, if necessary, the username (User) and the password (Password). At start of opsi-wmi-test.exe, some default values (Computer = localhost, NameSpace = \root\cimv2, User = < > , Password = < >) are given. These parameters allow the connection to the local WMI service and
access to the most used WMI classes. (Figure 9.1)
It is also possible to connect to remote WMI services of other computers in the network by using network name, username and password, respectively.

Figure 1. GUI of opsi-wmi-test. At the upper section (Connection to WMI service) you are asked to input the data which is necessary to connect to the WMI service. At the lower section (Request to WMI service), you are asked to input the data which is necessary to make the desired query to the WMI service.

At the lower section (Request to WMI service), you are asked to input the data which is necessary to make the desired query to the WMI service. You must input the WMI class or its alias as well as the properties you like to query for. You can input the WMI class or its alias directly in the respective field (Class or Alias) or you can select it from a list. (Figure 9.2). You should know that not all WMI Classes have Aliases. If the class you selected has one, its Alias will be automatically selected in the Alias field. If not, <No Alias> will be displayed. After the WMI class is given, its properties will be displayed in the field Available WMI Properties.
Available properties can be selected and moved into the field Selected WMI Properties using drag & drop or by clicking on the button (Figure 9.3). Note: This works also the other way around. Properties can be moved from Selected WMI Properties back to Available WMI Properties via drag & drop or the button.
Figure 2. Available properties can be selected and moved into the field Selected WMI Properties using drag & drop or by clicking on the button \([\rightarrow]\). Note: This works also the other way around. Properties can be moved from Selected WMI Properties back to Available WMI Properties via drag & drop or the button \([\leftarrow]\).
Figure 3. After moving the desired properties to the field Selected WMI Properties click on the Button Execute to send the query to the WMI service. You can specify your query within the field Condition using SQL statements.

After moving the desired properties to the field Selected WMI Properties click on the Button Execute to send the query to the WMI service. You can specify your query within the field Condition using SQL statements (example: \texttt{where} \texttt{<property> = <value>}).

The final query will be dispalyed in the Query field. It can be copied but not modified there. The result of your query is displayed in a second window (figure 9.4). If the query could not be processed the result is an error message.
Figure 4. The result of the query is displayed in a second window.

In a third window, you will have all the data about your query: Namespace, Selected Class, Properties list, Selected Properties and All available Properties. You can use all those fields for copying and pasting in your opsi script.
Figure 5. In a third window, you will have all the data about your query: Namespace, Selected Class, Properties list, Selected Properties and All available Properties.

9.4.2. Producing String Lists from Strings [W/L/M]

* createStringList (<string0>, <string1>, ... ) : stringlist [W/L/M]
form a String list from the values of the listed String expressions. For example, by

set $list1$ = createStringList ( 'a', 'b', 'c', 'd' )

we get a list of the first four letters of the alphabet.

The following two functions produce a String list by splitting some string: splitString (<string1>, <string2>) : stringlist [W/L/M]
generates the list of partial strings of <string1> (including empty strings) before resp. between the occurrences of <string2>. E.g.,
defines the list 
"", "", "server", "share", "directory"
If the given string is in the list of confidential strings, so the resulting string parts will also be added to the list of confidential strings.

• \texttt{splitStringOnWhiteSpace (<string>) \rightarrow stringlist [W/L/M]}
  slices StringVal by the "white spots" in it. E. g.

\begin{verbatim}
set $list1$ = splitStringOnWhiteSpace("Status   Lokal     Remote         Netzwerk")
\end{verbatim}

produces the list "Status", "Lokal", "Remote", "Netzwerk"
no matter how many blanks or tabs constitute the white space between the words.
If the given string is in the list of confidential strings, so the resulting string parts will also be added to the list of confidential strings.

9.4.3. Loading Lines of a Text File into a String List

• \texttt{loadTextFile (<file name>) \rightarrow stringlist [W/L/M]}
  reads the file <file name> and generates the string list, that contains all lines of the file.

• \texttt{loadTextFileWithEncoding( <file name>, <encoding>) \rightarrow stringlist [W/L/M]}
  reads the file <file name> and generates the string list, that contains all lines of the file. The string will be reencoded from <encoding> to system encoding.

• \texttt{loadUnicodeTextFile (<file name>) \rightarrow stringlist [W/L/M]}
  reads the unicode text file <file name> and generates the string list, that contains all lines of the file. By this call, the strings are converted into the system default 8 bit code.

• \texttt{getSectionNames(<ini-file>) \rightarrow stringlist [W/L/M]}
  interprets the specified file as an inifile, looks for list of all lines of form 
  \{<SectionName>\}
  and returns the pure section names (without brackets).

• \texttt{GetSectionFromInifile(<ini-file-section>,<ini-file>) \rightarrow stringlist [W/L/M]}
  +interprets the specified file as an inifile, looks for the list related to the +<ini-file-section>+ chosen in input 
  +and returns the complete raw section content in a stringlist.

9.4.4. Simple String Values generated from String Lists or Files [W/L/M]

• \texttt{composeString (string list}, \texttt{<Link>) \rightarrow string [W/L/M]}
  With this function, the elements of any String list can be glued to one another, mediated by a "glue string".
  E. g. if $\texttt{list1}$ represents the list $a$, $b$, $c$, $d$, $e$
  by
$line$ = composeString ($list1$, " | ")

we assign the value "a | b | c | d | e" to $line$.

• takeString (<index>, <list>) : string [W/L/M]
  For example, if $list1$ represents the list of the first five letters of the alphabet, using

  takeString (2, $list1$)

we get string "c" (since list counting starts with 0).

Negative values of index go downwards from the list count value. E.g.,

  takeString (-1, $list1$)

return the last list element, that is "e".
see also : setStringInListAtIndex
see also : takeString

• takeFirstStringContaining(<list>,<search string>) : string [W/L/M]
  returns the first string of the list which contains the <search string>.
  Returns an empty string if no matching string was found.

• getValue(<key string>, <hash string list>) : string [W/L/M]
  This function tries to interpret a String list as list of lines of the form key=value
  It looks for the first line, where the string <key> is followed by the equality sign, and returns the remainder of the line (the value, the
  string that starts after the equality sign). If there is no fitting line, it returns the string NULL.
  The function is required for using the GetLocaleInfoMap and getFileVersionMap string list functions (cf. Section 9.4.1, “Info
  Maps”).

• getValueBySeparator(<key string>,<separator string>,<hash string list>) : string //since 4.11.2.1 [W/L/M]
  works like getValue but you have to give the <separator string> so that can also work with hashes like
  key:value

• getValueFromFile(<key string>, <file name>) : string //since 4.11.4.4 [W/L/M]
  Searches in <file name> for a key/value pair with key <key string> and separator string = and returns the value. If <key string> is not
  found it returns an empty string.

• getValueFromFileBySeparator(<key string>,<separator string>,<file name>) : string //since 4.11.4.4 [W/L/M]
  Searches in <file name> for a key/value pair with key <key string> and separator string <separator string> and returns the value. If <key
  string> is not found it returns an empty string.

• count (<list>) : string (number) [W/L/M]
  returns the number of elements of the string list <list> as string.
  e.g. for $list1$ composed as
  a, b, c, d, e
  count ($list1$) has the value "5".
9.4.5. Producing String Lists from opsi-script Sections [W/L/M]

* **retrieveSection (<section name>): stringlist** [W/L/M]
  gives the lines of the specified section as string list.

* **getOutStreamFromSection (<dos section name>): stringlist (output)** [W/L/M]
  invokes the section and – at this moment implemented only for DosInAnIcon (ShellInAnIcon), ExecWith and ExecPython calls – captures the output to standard out and standard error of the invoked commands writing them into a string list. For example:

```plaintext
set $list$ = getOutStreamFromSection ('DosInAnIcon_netstart')

lis$1$ contains among some surrounding stuff the list of all mounted shares of a PC.
```

see also: *getReturnListFromSection*

see also: *executeSection*

There are 3 shortcuts for simple calls to the shell. At Windows these commands runs in the sysnative mode.

* **shellCall (<command string>): stringlist (output)** [since 4.11.4.2] [W/L/M]
  Executing <command string> with the standard shell (cmd.exe / bash)

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

Is a shortcut for this expression:

```plaintext
DosInAnIcon_netstart winst /sysnative
```

see also: *shellCall_list*

* **shellCall (<command string>): noresult** [since 4.11.6.1] [W/L/M]

```plaintext
shellCall('net start')
```

Is a shortcut for this expression:

```plaintext
DosInAnIcon_netstart winst /sysnative
```

see also: *shellCall*

* **shellCall (<command string>): string (exitcode)** [since 4.11.6.1] [W/L/M]

```plaintext
set $exitcode$ = shellCall('net start')
```

Is a shortcut for this expression:
see also: shellCall_str

- `getReturnListFromSection ( section name ) [W/L/M]`
  For some section types - at this moment implemented only for XMLPatch sections and opsiServiceCall sections - there is a specific `return` statement which yields some result of the execution of the section (assumed to be of String list type).
  E.g. we may use the statement

```plaintext
set list1 = getReturnListFromSection ('XMLPatch_mime "c:\mimetypes.rdf"')
```

to get a specific knot list of the XML file `mimetypes.rdf`. (More info to XMLPatch sections at Section 10.9, “XMLPatch Sections [W]” in this manual).
Or the list of opsi clients is produced by the reference to a opsi service call:

```plaintext
DefStringList $result$
set $result$=getReturnListFromSection("opsiservicecall_clientIdsList")
[opsiservicecall_clientIdsList]
  "method"="getClientIds_list"
  "params":[]
```

see also: getOutStreamFromSection see also: executeSection

### 9.4.6. Stringlists from the registry [W]

- `getRegistryKeyList32(<regkey>) : stringlist [W]`
  Returns a stringlist with the names of all keys within `<regkey>`.
  32 Bit mode (with redirection). Available since 4.11.3

- `getRegistryKeyList64(<regkey>) : stringlist`
  Returns a stringlist with the names of all keys within `<regkey>`.
  64 Bit mode (without redirection). Available since 4.11.3

- `getRegistryKeyListSysnative(<regkey>) : stringlist`
  Returns a stringlist with the names of all keys within `<regkey>`.
  Mode (redirection) depends on the architecture of the OS. Available since 4.11.3

- `getRegistryVarList32(<regkey>) : stringlist`
  Returns a stringlist with the names of all value entries associated with key `<regkey>`.
  32 Bit mode (with redirection). Available since 4.11.3

- `getRegistryVarList64(<regkey>) : stringlist`
  Returns a stringlist with the names of all value entries associated with key `<regkey>`.
  64 Bit mode (without redirection). Available since 4.11.3
• `getRegistryVarListSysnative(<regkey>) : stringlist`
  Returns a stringlist with the names of all value entries associated with key `<regkey>.
  Mode (redirection) depends on the architecture of the OS. Available since 4.11.3

• `getRegistryVarMap32(<regkey>) : stringlist`
  Provides a map of all name=value pairs in the given registry key `<regkey>.
  32 Bit Mode (with redirection). Since 4.11.3

• `getRegistryVarMap64(<regkey>) : stringlist`
  Provides a map of all name=value pairs in the given registry key `<regkey>.
  64 Bit Mode (with redirection). Since 4.11.3

• `getRegistryVarMapSysnative(<regkey>) : stringlist`
  Provides a map of all name=value pairs in the given registry key `<regkey>.
  Mode depend on the architecture of the operating system. Since 4.11.3

Example:
At first, we create entries in the registry with the following example code:

```
Registry_createkeys /32Bit

[Registry_createkeys]
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]
```

Given the registry entries in the example above, and the following code:

```
set $list$ = getRegistryVarList32("hklm\software\opsi.org\opsi-script-test")
```

we will see the following values in the 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
```

Then we call:

```
set $list$ = getRegistryVarMap32("hklm\software\opsi.org\opsi-script-test")
```

The following Log:
Given the registry entries in the example above, and the following code:

```plaintext
set $list$ = getRegistryKeyList32("hklm\software\opsi.org\opsi-script-test")
```

we will get the following key in the log:

```plaintext
set $list$ = getRegistryKeyList32("hklm\software\opsi.org\opsi-script-test")
```

9.4.7. Stringlists from the Product Properties [W/L/M]

- **getProductPropertyList**(<propname>,<default value>) [W/L/M] // seit 4.11.3
  returns a stringlist of values that are referred to by the multivalue product property <propname>. If you call the function `GetProductProperty` with a multivalue property, then you will get the selected values in a comma-separated string format. This will lead to problems if the returned values contain any comma chars that are not meant to be parsed. If there is no connection to the opsi server, then the resulting stringlist is constructed in the following way:
  Since 4.12.4.32 it is checked if there is a file `properties.conf` in the `ScriptPath` directory. If this file is found we try to get the value from this file. The file will be interpreted as list of key=value pairs. In case of a stringlist the entry should have the pattern `<property name>=<list value>` for example: `myproperty=["entry1","entry2","entry3"]`. If there is no file `properties.conf` or it does not contain the searched entry, so value `<default value>` is returned. `<default value>` describes the return value if no connection to the opsi-server is available. If `<default value>` is a string expression this string is the first element of the returned list. Since 4.11.5.6 `<default value>` may also be a string list expression. Since 4.12.4.32 `<default value>` may also be a string list from the kind: `['ab','cd','de']`

Example:

```plaintext
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",$propertyList$)
;;;;; since 4.12.4.32 also possible:
Set $list$ = GetProductPropertyList ("dummymulti",’"ab","cd","de"’)"
Since 4.12.4.32: if the file `%ScriptPath%\properties.conf` exists with the content:

```
propstr = from file
proplist = ["from file", "huhu"]
```

the following script will be (without opsi service) successful:

```
[Actions]
DefStringList $list$
DefVar $str$
set $str$ = GetProductProperty('propstr','')
if $str$ = "from file"
  comment "got it"
else
  comment "failed"
endif
set $list$ = GetProductPropertyList('proplist','')
if takeString(0,$list$) = "from file"
  comment "got it"
else
  comment "failed"
endif
```

see also: [asConfidential_list]

### 9.4.8. Other String Lists [W/L/M]

- **getProfilesDirList : stringlist** //since 4.11.3.2 [W/L/M]
  Provides a list of paths to the local profiles.
  [W]: Profiles that contain the following words will **not** be considered:
  - `localservice`
  - `networkservice`
  - `systemprofile`
  The profile of `Default Users` is included in the list.
  **All User** or `Public` are not included in the list.
  
  [L]: You get a list of the existing user directories from all users with a UID >= 1000.

  Example:

  ```
  set $list1$ = getProfilesDirList
  ```

  results in the following 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 : stringlist** //since 4.11.1.2; gives list of exename;pid;dom/user [W/L/M]
  Provides a list of running processes.
  For each process you get one line with a ; separated list of the following process information:
  
  - [W]: `Name of running exe`
  - [L]: short name of running process
  - [W/L/M]: `PID`
- [W]: Domain/User. [L]: User
- [L]: full command line of the process

* listFiles (<Path>, <Searchmask>, <SearchSubDirectories>, [<Redirection>]) : stringlist [W/L/M]

Provides a stringlist of all files found in <Path> (e.g. "C:\Windows\system32") which fits to the given <Searchmask> (e.g. "*.dll"). If <SearchSubDirectories> is set to "True" then all subdirectories within the given <Path> are included in the search. Setting <SearchSubDirectories> to "False" excludes the subdirectories from the search.

[W] Optional the parameter <Redirection> can be set to "64bit" or "SysNative". This enables to search in (system) directories which otherwise are not searchable because of redirection of 32-bit programs at 64-bit systems (only 64bit Windows). See also 64 Bit Support on Windows [W]

example:

```
message "Test of function listFiles"
DefVar $Path$
DefStringList $Files$
Set $Path$ = "%System%"
Set $Files$ = listFiles($Path$,"*.Devices.*.dll","False")
```

results in the following log:

```
message Test of function listFiles
(created string list $Files$)
The value of the variable "$Path$" is now: "C:\Windows\system32"
The value of the variable "$Files$" is now:
]string 0)C:\Windows\system32\Windows.Devices.AllJoyn.dll
]string 1)C:\Windows\system32\Windows.Devices.Background.dll
]string 2)C:\Windows\system32\Windows.Devices.Background.ps.dll
]string 4)C:\Windows\system32\Windows.Devices.Custom.dll
]string 5)C:\Windows\system32\Windows.Devices.Custom.ps.dll
]string 6)C:\Windows\system32\Windows.Devices.Enumeration.dll
]string 7)C:\Windows\system32\Windows.Devices.Haptics.dll
]string 9)C:\Windows\system32\Windows.Devices.Lights.dll
]string 10)C:\Windows\system32\Windows.Devices.LowLevel.dll
]string 18)C:\Windows\system32\Windows.Devices.Radios.dll
]string 20)C:\Windows\system32\Windows.Devices.SmartCards.dll
]string 22)C:\Windows\system32\Windows.Devices.SmartCards.Phone.dll
]string 23)C:\Windows\system32\Windows.Devices.Wifi.dll
]string 26)C:\Windows\system32\Windows.Devices.WifiDirect.dll
```

* replaceOpsiConstants(<string list>) : stringlist //since 4.12.3.6 [W/L/M] returns a list, which has all occurrences of opsi constants in <string list> replaced with their value.

see also: replaceOpsiConstants (string)

9.4.9. Transforming String Lists [W/L/M]
• **getSubList** (<start index>: <end index>, <list>) : stringlist [W/L/M]
  
  returns a partial list of a given list.
  
  E.g., if list represents the list of letters a, b, c, d, e, by the statement:

  ```
  set $list1$ = getSubList(1 : 3, $list$)
  ```

  we get the partial list b, c, d. Begin index as well as end index have to be interpreted as the index of the first and last included list elements. The counting starts with 0.

  Default start index is 0, default end index is the index of the last element of the list.

  Therefore, (for the above defined list1) the command

  ```
  set $list1$ = getSubList(1 : , $list$)
  ```

  yields the list b, c, d, e.

  ```
  set $list1$ = getSubList(1 , $list$)
  ```

  produces a copy of the original list.

  It is possible to count backwards in order to determine the last index:

  ```
  set $list1$ = getSubList(1 : -1, $list$)
  ```

  defines the list of elements starting with the first and ending with the last element of the list – in the above example we again get list b, c, d, e.

  ```
  set $list1$ = getSubList(1 : -2, $list$)
  ```

  defines the list of elements starting with the first and ending with the second to last element of the list – in the above example we get list b, c, d.

  Since version 4.12.0.35 you may beside numbers also use string expressions: strings, string variables or string functions.

  ```
  set $tmp1$ = "1"
  set $tmp2$ = "3"
  set $list1$ = getSubList( $tmp1$ : $tmp2$ , $list1$)
  ```

  ```
  set $list2$ = createStringList("","-1","0","1","2","3","4","5","6",)
  ```

  ```
  set $list1$ = getSubList(takestring(3,$list2$):takestring(5,$list2$), $list1$)
  ```

  • **getListContaining** (<list>,<search string>) : stringlist [W/L/M]

  returns the first string from <list> which contains <search string>. Returns empty string if <search string> is not found.

  • **getListContainingList** (<list1>,<list2>) : stringlist //since 4.11.3.7 [W/L/M]

  returns the intersection of list1 and list2.

  • **getSubListByMatch** (<search string>, <target list>) :stringlist //since 4.12.0.14 [W/L/M]

  returns the part of <target list> where the string matches with <search string>. The check is performed case-insensitive.

  • **getSubListByMatch** (<search list>, <target list>) :stringlist //since 4.12.0.14 [W/L/M]

  returns the part of <target list> where the string matches with one of the strings of <search list>. The check is performed case-insensitive.
• getSubListByContaining (<search string>, <target list>):stringlist //since 4.12.0.14 [W/L/M]
  returns the part of <target list> where the string contains <search string>.
The check is performed case-insensitive.

• getSubListByContaining (<search list>, <target list>):stringlist //since 4.12.0.14 [W/L/M]
  returns the part of <target list> where the string contains with one of the strings of <search list>.
The check is performed case-insensitive.

• getSubListByKey (<search string>, <target list>):stringlist //since 4.12.0.14 [W/L/M]
  returns the part of <target list> where the string starts with <search string>=.
The check is performed case-insensitive.

• getSubListByKey (<search list>, <target list>):stringlist //since 4.12.0.14 [W/L/M]
  returns the part of the key/value <target list> where the key is one of the strings of <search list>.
The check is performed case-insensitive.

• getKeyList (<list>):stringlist //since 4.12.0.14 [W/L/M]
  returns from the the key/value list <list> (in the format key=value) the list of keys.
  Is an entry in <list> not in the format key=value, the complete string will be part of the result list.

• takeFirstStringContaining(<list>,<search string>) : string [W/L/M]
  returns the first string from <list> which contains <search string>.
  Return en empty string if <search string> is not found.
  see also : [takeFirstStringContaining]

• addtolist(<list>,<string>) : stringlist //since 4.10.8 [W/L/M]
  Appends <string> to the list <list>.

• addListToList(<dest list>,<src list>) : stringlist //since 4.10.8 [W/L/M]
  Appends the list <list2> to the list <list1>.

• reverse (<list>) : stringlist [W/L/M]
  produces the inverted list,
  if $list$ is a, b, c, d, e, by

  ```
  set $list1$ = reverse ($list$)
  ```

  we get the $list1$ e, d, c, b, a.

• emptylist (<list>) : stringlist //since 4.11.3.7 [W/L/M]
  clears the list.
• **reencodeList**(<list>, <from>, <to>) : stringlist //since 4.11.4.2 [W/L/M]
  assumes that <list> is encoded in <from> and returns the in <to> encoded version of <list>. <from> and <to> are encodings as listet in chapter opsi-script encoding.

• **removeFromListByContaining**(<search string>, <target list>) : stringlist //since 4.11.5.1 [W/L/M]
  Returns a copy of <target list> where all lines that contains <search string> are removed. The match to <search string> is case insensitiv.

• **removeFromListByContaining**(<search list>, <target list>) : stringlist //since 4.11.5.1 [W/L/M]
  Returns a copy of <target list> where all lines are removed that contains a string out of <search list>. The match to <search list> is case insensitiv.

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:

```plaintext
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 ($ConstTest$ = $CompValue$)
    set $TestResult$ = "passed"
    LogWarning "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 ($ConstTest$ = $CompValue$)
    set $TestResult$ = "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $ConstTest$ = "0"
set $list2$ = removeFromListByContaining("line", $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
    set $TestResult$ = "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 ($ConstTest$ = $CompValue$)
    set $TestResult$ = "passed"
```
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif

comment "search with string function"
set $ConstTest$ = "$\star$"
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 $ConstTest$ = "$\star$"
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"

remFromListByMatch(<searchstring>,<target list>) : stringlist //since 4.11.6 [W/L/M]
Returns a copy of <target list> where all lines are removed that exactly match a line out of <search list>. The match to <search string> is case insensitive.
see also : [removeFromListByContaining_str]
see also : [removeFromListByContaining_list]

setStringInListAtIndex(<newstring>,<list>,<indexstr>) : stringlist //since 4.11.6 [W/L/M]
Replaces in the existing stringlist <list> the existing string at <index> by <newstring>. If <index> is to hight, <newstring> will be appended. In case of an error the result is an empty string list.
see also : [takeString]

areListsEqual(<strlist1>, <strlist2>, <flag>) : boolean
Evaluates if the 2 stringlists <strlist1> and <strlist2> are equal according to the chosen flag:
If flag=FLAG_AUTOMODE: The function will do a case insensitive comparaison. It handles key/value stringlists (with the structure : key1=value1) or simple stringlists.
If flag=FLAG_AUTO_CS : The function will do a case sensitive comparaison. It also handles both key/value stringlists (with the structure : key1=value1) or simple stringlists.
If flag=FLAG_STRING_CS : The function will do a case sensitive comparaison that only handles simple stringlists (So it does not take into consideration the key/value structure).
9.4.10. Iterating through String Lists [W/L/M]

An important usage of string lists is based on the possibility that the script runs through all elements of a list executing some operation on each string element.

The syntax to define this repetition is:

```
• for %s% in <list> do <one statement | sub section>
```

This expression locally defines a string variable %s% that takes one by one the values of the list elements. <one statement> can be any single statement that can exist in a primary section or (and most interestingly) it may be a subsection call. The locally defined iteration index %s% exists in the whole context of statement, in particular in the subsection if statement is a subsection call.

The replacement mechanism for %s% always works like that for constants: The name of the variable is replaced by the element values. If we iterate through a list a, b, c and the iteration index is named %s%, we get for %s% one by one a, b, c – not the String values. To reproduce the original list elements we have to enclose %s% in citation marks.

Example: Let $list1$ be the list a, b, c, d, e, and $line$ a String variable. The statement

```
for %s% in $list1$ do set $line$ = $line$ + "%s%"
```

iterates through the list elements internally executing

```
$line$ = $line$ + "a"
$line$ = $line$ + "b"
$line$ = $line$ + "c"
$line$ = $line$ + "d"
$line$ = $line$ + "e"
```

Such, finally line has value abcde. If we omitted the citation marks around %s% we would get a syntax error for each iteration step.

Please note: The note variable is only valid in the directly called procedure. If it is needed in sub programs of it its value must be transferred to a global variable.

9.5. opsiservicecall and json Related functions [W/L/M]

This functions may be used to analyze and modify JSON expressions. In order to work with these functions, you have to understand the structure of the used JSON expression. These function can help to use the opsiservicecall sections.

```
• jsonIsValid(<jsonstr> ) : boolean //since 4.11.6: [W/L/M]
  returns true if <jsonstr> is a valid JSON expression.

• jsonIsArray(<jsonstr> ) : boolean //since 4.11.6: [W/L/M]
  returns true if <jsonstr> is a valid JSON array.

• jsonIsObject(<jsonstr> ) : boolean //since 4.11.6: [W/L/M]
  returns true if <jsonstr> is a valid JSON object.
```
• \texttt{jsonAsObjectHasKey(<jsonstr>,<keystr>) : boolean \ since 4.11.6 [W/L/M]}
  returns true if <jsonstr> is a valid JSON object, which contains <keystr> as key.
  The following example returns true:

  \[
  \text{jsonAsObjectHasKey('"productVersion" : "4.4.1","packageVersion" : "2","productId" : "jedit"'),'productId'}\]

• \texttt{jsonAsArrayCountElements(<jsonstr>) : intstr \ since 4.11.6 [W/L/M]}
  if <jsonstr> is a valid JSON array, the function returns the number of elements in the array.
  In case of an error, the result is "0"

• \texttt{jsonAsObjectCountElements(<jsonstr>) : intstr \ since 4.11.6 [W/L/M]}
  if <jsonstr> is a valid JSON object, the function returns the number of elements in the object.
  In case of an error, the result is "0"

• \texttt{jsonAsArrayGetElementByIndex(<jsonstr>, <indexstr>) : jsonstring \ since 4.11.6 [W/L/M]}
  if <jsonstr> is a valid JSON array, the function returns the element at the index <indexstr>.
  The index starts with "0".
  In case of an error, the result is ""

• \texttt{jsonAsObjectGetValueByKey(<jsonstr>, <keystr>) : valuestring \ since 4.11.6 [W/L/M]}
  returns from the JSON object <jsonstr>, the value of the key <keystr>.
  In case of an error, the result is ""

• \texttt{jsonAsObjectSetValueByKey(<jsonstr>, <keystr>,<valuestring>) : jsonstring \ since 4.11.6 [W/L/M]}
  returns a string with a JSON object. This object is the given <jsonstr>, where the value of the key <keystr> is set to <valuestring>. If the key <keystr> did not exist, it will be created.
  If you are sure that you want to create <valuestring> as string value (and not as number or boolean, ...) , than we recommend to use the function: \texttt{jsonAsObjectSetStringValueByKey}.
  In case of an error, the result is ""

• \texttt{jsonAsObjectSetStringValueByKey(<jsonstr>, <keystr>,<valuestring>) : jsonstring \ since 4.11.6 [W/L/M]}
  returns a string with a JSON object. This object is the given <jsonstr>, where the value of the key <keystr> is set to <valuestring> as a string (quoted). If the key <keystr> did not exist, it will be created.
  If you are not want to create <valuestring> as string value (but as number or boolean, ...) , than we recommend to use the function: \texttt{jsonAsObjectSetValueByKey}.
  In case of an error, the result is ""

• \texttt{jsonAsObjectDeleteByKey(<jsonstr>, <keystr>) : jsonstring \ since 4.11.6.4 [W/L/M]}
  returns a string with a JSON object. This object is the given <jsonstr>, where the value of the key - value pair with the key <keystr> is removed.

• \texttt{jsonAsArrayPutObjectByIndex(<jsonstr>, <indexstr>, <objectstr>) : jsonstring \ since 4.11.6 [W/L/M]}
  returns a string with a JSON array. This array is the given <jsonstr>, where at the index <indexstr> the object <objectstr> is set.
  In case of an error, the result is ""
• **jsonAsArrayDeleteObjectByIndex**(<jsonstr>, <indexstr>) : **jsonstring**  //since 4.11.6.4: [W/L/M]
  returns a string with a JSON array. This array is the given `<jsonstr>`, where the object at the index `<indexstr>` is removed.
  In case of an error, the result is ""

• **jsonAsArrayToStringList**(<jsonstr>) : **stringlist**  //since 4.11.6: [W/L/M]
  returns a stringlist that contains the elements of the given JSON array `<jsonstr>`, with one element per line.

• **jsonStringListToJsonArray**(<strlist>) : **jsonstr**  //since 4.11.6: [W/L/M]
  returns a string that contains a JSON array that is constructed from the given string list `<strlist>`, where every line of this list is handled as one array element.

• **jsonObjectGetKeyList**(<jsonstr>) : **stringlist**  //since 4.11.6: [W/L/M]
  returns the list of keys that are found in the JSON object `<jsonstr>`.

Example: Restoring productOnClient entries from a file to the server:
9.6. Calculating with numbers [W/L/M]

opsi-script scripts do not have a special type of variables for numbers. But there are some functions to help calculating with numbers.
• calculate(<arithmetic string expression>) : string (number)
  this string function calculates the arithmetic expression of the string <str> and returns the rounded result as a string.
  Internally the calculations are done with real numbers. This function accepts the operators +, -, *, / and round brackets ( ).
  In case of an error, an empty string is returned and the error counter is incremented. If the passed string contains any characters other than numbers, valid operators and brackets, this results in an error.
  If the second operand is missing, the first operand is also taken as the second operand and vice versa: 5+ = 10 ; 5* = 25. So the strings that are used to assemble the argument should be validated by the function isNumber.
  (since version 4.11.3.5)
  see also : [isNumber]

Example:

```
set $ConstTest$ = "0"
set $CompValue$ = calculate("-1+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 $string2$ = "0"
set $ConstTest$ = ""
comment " expecting division 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
```
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 $ConstTest$ = "100"
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 $string2$ = "ten"
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 $string2$ = "ten"
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

For more examples refer to the product opsi-script-test at the section $Flag_calculate$ = "on"

There is as well a comparison expression for comparing Strings as (integer) numbers. If any of them cannot be converted to a number an error will be indicated.

This number comparison expression has the same form as the String comparison but for an INT prefix of the comparison sign:
Such, we can build expressions as

```plaintext
if $Name1$ INT<= $Name2$
```

or

```plaintext
if $Number1$ INT>= $Number2$
```

see also: Boolean Expressions

* isNumber(<str>) : bool //since 4.11.3: true if <str> represents an integer [W/L/M]
  Boolean function which returns true if <str> represents an integer.
  see also: [calculate]

## 9.7. XML2 Functions [W/L/M]

These XML2 section and functions replaces the older and Windows only XMLPatch section XMLPatch Sections. If you find some problems or have need for additional features, do not hesitate to contact us.

For the wording in this chapter see the chapter XML2 Sections / XML structure and wording XML structure and wording

### Encoding

XML files are normally encoded in UTF-8. So the XML2 functions expect the given XML files in UTF-8. The XML2 Section also expect the encoding of the given xml file should be UTF-8 by default but here we have the possibility to work also with different encodings.

#### getXml2DocumentFromFile(<path to xml file>) : xml2stringlist //since 4.12.1

Reads the xml file <path to xml file> and returns a string list that may be used in other xml2 functions as <xml2stringlist>

For example see XML2 Sections / Examples XML2 Examples

#### getXml2Document(<stringlist with xml>) : xml2stringlist //since 4.12.1

If <stringlist with xml> contains valid xml data the function returns a string list that may be used in other xml2 functions as <xml2stringlist>

#### xml2GetFirstChildNodeByName(<xml2stringlist>, <node name str>) : xml2stringlist //since 4.12.1

Searches in the given <xml2stringlist> the first occurrence of a node with the name <node name str> and returns this node (with all child nodes) as a xml2stringlist.

For example see XML2 Sections / Examples XML2 Examples

#### getXml2UniqueChildnodeByName(<xml2stringlist>, <node name str>) : xml2stringlist //since 4.12.1.

Searches in the given <xml2stringlist> the occurrences of a nodes with the name <node name str> and if there is only one, this one is returned (with all child nodes) as a xml2stringlist.

If there is none or more than one child found, a empty string list is returned.

#### getXml2AttributeValueByKey(<xml2stringlist>, <attr name str>) : string //since 4.12.1.

Searches in the given <xml2stringlist> the first node and in this node an attribute with the name <attr name str> and returns the value of this
attribute.
If not found or on any other error, an empty string will be returned.
For example see XML2 Sections / Examples XML2 Examples

```
getXml2Text( <xml2stringlist> ) : string //since 4.12.1.
```
Returns the xml data given by `<xml2stringlist>` as a single line string.

see also : XML related functions (XML2)
see also : XML2 Section

### 9.8. Regular expression related functions [W/L/M]

> Working with regular expressions has some disadvantages:
> Finding the correct regular expression can be difficult. So use tools to test your expressions.
> Code written with regular expressions is hard to read.
> Make comments in your code that explain what you try to do.

There are different kinds of regular expressions: Perl, Javascript, Java, …
The kind of regular expression implemented here is a variant of the perl style or PCRE.
A detailed documentation of the here used variant you will find at:

> You should use a tool to test your regular expression.
> We recommend the opsi-regexpr-tester.exe application. You can download it here:
> https://download.uib.de/opsi4.2/misc/helper/opsiRegExprTest.exe

* opsi-regexpr-tester.exe

This application (opsi-regexpr-tester.exe) helps you test your Regular Expression.
The user interface of `opsi-regexpr-tester` offers you two fields: First field (`Regular Expression:`) is where you are asked to input the Regular Expression you want to test. Second field (`Text:`) is where you should input the text on which you want to test your regular expression. Then, you can insert Pattern Modifiers Flags, explained in details in the documentation previously mentioned:


We propose 6 flags:

- i, case-insensitive
- m, multi-line strings
- s, single line strings
- g, greediness
- x, eXtended syntax
- r, Russian ranges

You can select the ones you want to insert by checking the corresponding box. They will be automatically added at the beginning of the `Regular Expression:` field. Please note that the flags are always put before the corresponding regular expression part!

Finally, you can click on the button "Examine" to test your Regular Expression.
If your regular expression matches the text or part of it, a successful result will appear, and the matches will be coloured in green on your text.
If your regular expression does not match the text or any part of it, a failure text result will appear.
You can clear both the Regular Expression and the Text fields by clicking on the button "Clear".

* `isRegexMatch(<string>, <pattern>) : boolean`
  It will return `true` if the pattern matches the string and return `false` if pattern does not matches the string, where pattern is the regular expression and the string would be searched for the matches.

Example:

```plaintext
comment "Testing with matching string"
set $ConstTest$ = "true"
set $CompValue$ = booltostring(isRegexMatch('abc efg', '.\abc.*'))
if ($ConstTest$ = $CompValue$
  comment "isRegexMatch passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "testing isRegexMatch failed"
endif

comment "Testing with non matching string"
set $ConstTest$ = "false"
set $CompValue$ = booltostring(isRegexMatch('abc efg', '.\xyz.*'))
if ($ConstTest$ = $CompValue$
  comment "isRegexMatch passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "testing isRegexMatch failed"
endif
```
• `getSubListByContainingRegex(<pattern>, <target list>)` : `stringlist`
  It will extract a sublist from target list by matching a single pattern, where sublist is a list of whole lines containing the pattern.

• `getSubListByContainingRegex(<pattern list>, <target list>)` : `stringlist`
  It will extract a sublist from target list by matching a list of patterns, where sublist is a list of whole lines containing any of the patterns in the pattern list.

Example:

```plaintext
comment "Testing with a single pattern"
set $string1$ = "\w+(\[\w-]+)*@[\[\w-]+\]\.[a-zA-Z]{2,6}"
set $list1$ = createStringList('uib gmbh','example@xyz.com and example2@xyz.com', 'client')
set $ConstTest$ = "example@xyz.com and example2@xyz.com"
set $list2$ = getSubListByContainingRegex($string1$, $list1$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
  comment "getSubListByContainingRegex passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "testing getSubListByContainingRegex failed"
endif

comment "Testing with a list of patterns"
set $list3$ = createStringList('\w+(\[\w-]+)*@[\[\w-]+\]\.[a-zA-Z]{2,6}', '.*uib')
set $ConstTest$ = "uib gmbh | example@xyz.com and example2@xyz.com"
set $list2$ = getSubListByContainingRegex($list3$, $list1$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
  comment "getSubListByContainingRegex passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "testing getSubListByContainingRegex failed"
endif
```

• `getRegexMatchList(<pattern>, <target list>)` : `stringlist`
  It will extract a sublist from the target list by matching a single pattern, where sublist is a list of exact matches for the pattern.

• `getRegexMatchList(<pattern list>, <target list>)` : `stringlist`
  It will extract a sublist from the target list by matching a list of patterns, where sublist is a list of exact matches for any of the patterns in the pattern list.

Example:
comment "Testing with a single pattern"
set $string1$ = "\w+([\w-]*@([\w-]+[.])+[a-zA-Z]{2,6})"
set $list1$ = createStringList('uib gmbh','client','example@xyz.com and example2@xyz.com'
set $ConstTest$ = "example@xyz.com | example2@xyz.com"
set $list2$ = getRegexMatchList($string1$, $list1$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
    comment "getRegexMatchList passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing getRegexMatchList failed"
endif

custom "Testing with a list of patterns"
set $list3$ = createStringList('\w+([\w-]*@([\w-]+[.])+[a-zA-Z]{2,6})','.*uib')
set $ConstTest$ = "uib | example@xyz.com | example2@xyz.com"
set $list2$ = getRegexMatchList($list3$, $list1$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
    comment "getRegexMatchList passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing getRegexMatchList failed"
endif

• removeFromListByContainingRegex(<pattern>, <target list>) : stringlist
  remove whole matching lines for a single pattern.

• removeFromListByContainingRegex(<pattern list>, <target list>) : stringlist
  remove whole matching lines for any of the patterns in the pattern list.

Example:

comment "Searching with a single expression"
set $string1$ = "\w+([\w-]*@([\w-]+[.])+[a-zA-Z]{2,6})"
set $list1$ = createStringList('uib gmbh','client','example@xyz.com and example2@xyz.com'
set $ConstTest$ = "uib gmbh | client"
set $list2$ = removeFromListByContainingRegex($string1$, $list1$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
    comment "removeFromListByContainingRegex passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing removeFromListByContainingRegex failed"
endif

custom "Searching with a list of expressions"
set $list3$ = createStringList('\w+([\w-]*@([\w-]+[.])+[a-zA-Z]{2,6})','.*uib')
set $ConstTest$ = "ui"
set $list2$ = removeFromListByContainingRegex($list3$, $list1$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
    comment "removeFromListByContainingRegex passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing removeFromListByContainingRegex failed"
endif

• stringReplaceRegex(<string>, <pattern>, <replacement string>) : string
  Replace exact matches in string with the replacement string.

Example:
set $ConstTest$ = "xyz abc gmbh"
set $CompValue$ = stringReplaceRegex('uib gmbh','.', 'xyz abc')
if ($ConstTest$ = $CompValue$)
    comment "stringReplaceRegex passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing stringReplaceRegex failed"
endif

• stringReplaceRegexInList(<target list>, <pattern>, <replacement string>) : stringlist
  Replace exact matches in the target list with the replacement string.

Example:

set $string1$ = "\w+(\w+\.)+@([\w-]+\.)+[a-zA-Z]{2,6}"
set $list1$ = createStringList('uib gmbh','client','example@xyz.com and example2@xyz.com')
set $ConstTest$ = 'uib gmbh | client | MATCH REMOVED and MATCH REMOVED'
set $list2$ = stringReplaceRegexInList($list1$, $string1$, $string2$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
    comment "stringReplaceRegexInList passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing stringReplaceRegexInList failed"
endif

For further examples see the product opsi-script-test especially the file sub-scripts/regex.opsiscript

9.9. URL related functions [W/L/M]

These functions can be used to analyze and create URLs.

They convert string URLs to a string list with the URL components:

- 'Protocol=proto'
- 'Username=usr'
- 'Password=pwd'
- 'Host=host'
- 'Port=8080'
- 'Path=/path/'
- 'Document=doc'
- 'Params=param'
- 'Bookmark=bookmark'

• parseUrl(<url string>) : stringlist
  Return url components as stringlist for the given <url string>.

Example:
comment "Testing parseUrl with all fields"
set $string1$ = "proto://usr:pwd@host:8080/path/doc?param#bookmark"
set $list1$ = createStringList('Protocol=proto', 'Username=usr', 'Password=pwd', 'Host=host', 'Port=8080', 'Path=/path/', 'Document=doc', 'Params=param', 'Bookmark=bookmark')
set $ConstTest$ = composeString ($list1$, " | ")
set $list2$ = parseUrl($string1$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
    comment "parseUrl passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing parseUrl failed"
endif

comment "Testing parseUrl with some fields"
set $string1$ = "ftp://example.abc.edu/"
set $list1$ = createStringList('Protocol=ftp', 'Username=', 'Password=', 'Host=example.abc.edu', 'Port=0', 'Path=/', 'Document=', 'Params=', 'Bookmark=')
set $ConstTest$ = composeString ($list1$, " | ")
set $list2$ = parseUrl($string1$)
set $CompValue$ = composeString ($list2$, " | ")
if ($ConstTest$ = $CompValue$)
    comment "parseUrl passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing parseUrl failed"
endif

* createUrl(<urlcomponents list>) : string
    return url string for the given urlcomponents.
    The <urlcomponents list> has not to be complete. You may give only the needed parts.

Example:

comment "Testing createUrl with all fields"
set $list1$ = createStringList('Protocol=proto', 'Username=usr', 'Password=pwd', 'Host=host', 'Port=8080', 'Path=/path/', 'Document=doc', 'Params=param', 'Bookmark=bookmark')
set $ConstTest$ = "proto://usr:pwd@host:8080/path/doc?param#bookmark"
set $CompValue$ = createUrl($list1$)
if ($ConstTest$ = $CompValue$)
    comment "createUrl passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing createUrl failed"
endif

comment "Testing createUrl with some fields"
set $list1$ = createStringList('Protocol=https', 'Host=www.example.com', 'Path=/b-c-d-330002341216/')
set $ConstTest$ = "https://www.example.com/b-c-d-330002341216/"
set $CompValue$ = createUrl($list1$)
if ($ConstTest$ = $CompValue$)
    comment "createUrl passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing createUrl failed"
endif

For further examples see the product opsi-script-test especially the file sub-scripts/urlparser.opsiscript

9.10. Network related functions [W/L/M]
isValidIP4 (<ip4adr>) : boolean
return true if the IPv4 address is valid.

Example:

```plaintext
comment "Testing with valid IPv4 address"
set $ConstTest$ = "true"
set $CompValue$ = booltostring(isValidIP4("255.255.0.0"))
if ($ConstTest$ = $CompValue$)
    comment "isValidIP4 passed"
else
    set $TestResult$ = "not o.k.".
    LogWarning "testing isValidIP4 failed"
endif

comment "Testing with invalid IPv4 address"
set $ConstTest$ = "false"
set $CompValue$ = booltostring(isValidIP4("255.256.0.0"))
if ($ConstTest$ = $CompValue$)
    comment "isValidIP4 passed"
else
    set $TestResult$ = "not o.k.".
    LogWarning "testing isValidIP4 failed"
endif
```

isValidIP4Network (<ip4adr>, <netmask>) : boolean
return true if the given ipv4 address is a valid network address.
Netmask can be given in both cidr and dotted-decimal notation.

Example:

```plaintext
comment "Testing with valid network address, where netmask is in cidr notation"
set $ConstTest$ = "true"
set $CompValue$ = booltostring(isValidIP4Network('192.168.0.0','/24'))
if ($ConstTest$ = $CompValue$)
    comment "isValidIP4Network passed"
else
    set $TestResult$ = "not o.k.".
    LogWarning "testing isValidIP4Network failed"
endif

comment "Testing with valid network address, where netmask is in dotted-decimal notation"
set $ConstTest$ = "true"
set $CompValue$ = booltostring(isValidIP4Network('192.168.0.0','255.255.255.0'))
if ($ConstTest$ = $CompValue$)
    comment "isValidIP4Network passed"
else
    set $TestResult$ = "not o.k.".
    LogWarning "testing isValidIP4Network failed"
endif

comment "Testing with invalid network address"
set $ConstTest$ = "false"
set $CompValue$ = booltostring(isValidIP4Network('198.51.100.223','/21'))
if ($ConstTest$ = $CompValue$)
    comment "isValidIP4Network passed"
else
    set $TestResult$ = "not o.k.".
    LogWarning "testing isValidIP4Network failed"
endif
```

isValidIP4Host (<ip4adr>, <netmask>) : boolean
return true if the given ipv4 address is a valid host address.
Netmask can be given in both cidr and dotted-decimal notation.

Example:
• `getIP4NetworkByAdrAndMask(<ip4adr>, <netmask>) : string`

    return network address for the given IP address and netmask.
    Netmask can be given in both cidr and dotted-decimal notation.

Example:

```plaintext
comment "Testing with netmask in cidr notation"
set $ConstTest$ = "198.48.0.0"
set $CompValue$ = getIP4NetworkByAdrAndMask('198.51.100.223', '12')
if ($ConstTest$ = $CompValue$)
    comment "getIP4NetworkByAdrAndMask passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing getIP4NetworkByAdrAndMask failed"
endif

comment "Testing with netmask in dotted decimal notation"
set $ConstTest$ = "198.48.0.0"
set $CompValue$ = getIP4NetworkByAdrAndMask('198.51.100.223', '255.240.0.0')
if ($ConstTest$ = $CompValue$)
    comment "getIP4NetworkByAdrAndMask passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing getIP4NetworkByAdrAndMask failed"
endif
```

• `getDefaultNetmaskByIP4adr (<ip4adr>) : string`

    return default netmask for the given IPv4 address

Example:

```plaintext
set $ConstTest$ = "255.255.0.0"
set $CompValue$ = getDefaultNetmaskByIP4adr("128.42.5.4")
if ($ConstTest$ = $CompValue$)
    comment "getDefaultNetmaskByIP4adr passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "testing getDefaultNetmaskByIP4adr failed"
endif
```
isValidFQDN (<domainName>) : boolean
returns true if the input string is a Fully Qualified Domain Name (FQDN) and false if not. A Fully Qualified Domain Name (FQDN) has the following conditions:

- Its total length should not exceed 254 characters,
- It should have at least 3 dot-separated labels,
- Each label’s length should not exceed 63 characters,
- Labels contain only letters, numbers and dashes,
- The last label (TLD) should contain only letters and should have at least 2 characters,
- Each label starts with a letter or a number (no dashes)

Example:

```plaintext
Message "Testing of isValidFQDN"
SetLogLevel=7
DefVar $TestResult$
DefVar $CompValue$
DefStringList $CorrectFQDNs$
DefStringList $IncorrectFQDNs$
Set $CorrectFQDNs$ = CreateStringList("www.uib.de", "opsi-script.uib.de", "m.opsi.org", "a-b-c.cc", "a-b-c.d.e.f.g.h.i.j.k.l.m", "a-b-c.d.e.f.g.h.i.j.k.l.m.n", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u.v", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u.v.w", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u.v.w.x", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u.v.w.x.y", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u.v.w.x.y.z")
Set $IncorrectFQDNs$ = CreateStringList("ui", "uib.de", "www.uib", "www.uib.d", "script.uib.de", "_script.uib.de", "www.uib.00", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u.v.w.x.y.z", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u.v.w.x.y.z", "a-b-c.d.e.f.g.h.i.j.k.l.m.n.o.p.q.r.s.t.u.v.w.x.y.z")
DefFunc myFQDNTester($expected$ : string, $fqdn$ : string, ref $TestResult$ : string) : void
  DefVar $CompValue$
  set $CompValue$ = booltoString(isValidFQDN($fqdn$))
  if $CompValue$ = $expected$
    comment "Testing isValidFQDN succeeded"
  else
    set $TestResult$ = "not o.k."
    LogWarning "Testing isValidFQDN failed"
  endif
endfunc
for %s% in $CorrectFQDNs$ do  myFQDNTester("true","%s%",$TestResult$)
for %s% in $IncorrectFQDNs$ do  myFQDNTester("false","%s%",$TestResult$)
```

For further examples see the product opsi-script-test especially the file sub-scripts/networkcalc.opsiscript

9.11. Process and Script Related functions [W/L/M]

- `waitForPackageLock(<seconds timeout string>,<bool should we kill>) : bool //since 4.11.6.1 [L]`
  Returns true if the Linux package system is not locked by an other process. If it is locked, it waits <seconds timeout string> to get the lock. If the timeout is reached and <bool should we kill> is true than the locking process is killed but using this feature is not recommended.

- `processIsRunning(<process name>) : boolean //since 4.11.6.1 [W/L/M]`
  Returns true if <process name> is found in the process list

isProcessChildOf(<searchprocstr>, <parentprocstr>): bool //since 4.12.4.35 [W/L/M]
Returns true if the process <searchprocstr> is a child of <parentprocstr>.
Beispiel:
if isProcessChildOf('opsiclientd.exe', 'opsiclientd.exe')
    comment "running in opsi service context"
endif

• shellCall (<command string>): stringlist (output) //since 4.11.4.2 [W/L/M]
  Executing <command string> with the standard shell (cmd.exe / bash)

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

Is a shortcut for this expression:

  set $list$ = getOutStreamFromSection ('DosInAnIcon_netstart winst /sysnative')

  [DosInAnIcon_netstart]
  net start

• shellCall (<command string>): noresult //since 4.11.6.1 [W/L/M]

  shellCall('net start')

Is a shortcut for this expression:

  DosInAnIcon_netstart winst /sysnative
  [DosInAnIcon_netstart]
  net start

• shellCall (<command string>): string (exitcode) //since 4.11.6.1 [W/L/M]

  set $exitcode$ = shellCall('net start')

Is a shortcut for this expression:

  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]
  powershellCall(<commandstr>, <access str>=sysnative [, <policy bool str>=true][, <optionstr> = "]")) : stringlist (output) //since 4.12.4.28 [W]
  Runs <command string> with the powershell.
  More exactly the powershell runs a script that looks like:

    trap { write-output $_ ; exit 1 }
    <commandstr>
    exit $LASTEXITCODE

The first line of the script makes sure that we get no exit code = 0 if the script fails with an exception. The last line gives the exit code of previous command back.

The architecture of the called powershell.exe is sysnative by default. Using the optional second parameter <access str> you may change
this default. In this case it has to be one of the following values: 32bit, sysnative, 64bit.
(see also: Chapter 64 Bit)
By Windows default the powershell has the execution policy Restricted which do not allow to run any unsigned scripts. In order to run
tasks the powershellCall function does by default the following: The current execution-policy is backedup and the execution-policy is set
to RemoteSigned. Then the script will be executed and finally the execution-policy is restored. This default behaviour may be switched off
by setting the optional third parameter <policy bool str> to “false”. If the powershellCall function is called where a stringlist is expected it returns a stringlist that contains the output of <commandstr>. Since 4.12.4.35:
If the ExecutionPolicy AllSigned is detected, the powershell will be modified from calling the temporary script by -File to -Command Get-
Content -Path <tempfilename> | Out-String | Invoke-Expression. In this mode the ExecutionPolicy is ignored. In this mode any given PASSPARAS are also ignored.
Since 4.12.4.28:
The optional <optionstr> may be used to pass additional modifiers to the function. Because the function powershellCall calls internal an
Execwith section, all possible Execwith modifier maybe also given in the <optionstr>. For details see here: Section 10.16.1, “Calling
parameters (Modifier)”.

Example:

```
set $list$= powershellCall('Get-Process -ProcessName "opsi*"')
```

Is a shortcut for this expression:

```
set $policy$ = takeString(0,shellCall('powershell.exe get-executionpolicy'))
shellCall('powershell.exe set-executionpolicy RemoteSigned')
set $list$ = getOutStreamFromSection ('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] [,<optionstr> = '']]) : noresult //since 4.12.0.16 [W]
powershellCall(<commandstr> [,<access str>=sysnative [,<policy bool str>=true] [,<optionstr> = '']]) : noresult //since 4.12.4.28
[W]
see [powershellCall_list]
The function powershellCall may also be called where no result is expected.

Example:

```
powershellCall('Get-Process -ProcessName "opsi*"')
```

• powershellCall(<commandstr> [,<access str>=sysnative [,<policy bool str>=true] [,<optionstr> = '']]) : string (exitcode) //since 4.12.0.16 [W]
powershellCall(<commandstr> [,<access str>=sysnative [,<policy bool str>=true] [,<optionstr> = '']]) : string (exitcode)
//since 4.12.4.28 [W]
see [powershellCall_list]
If the powershellCall function is called where a string is expected it returns a string that contains the exit code of the executed script.

Example:

```
set $exitcode$ = powershellCall('Get-Process -ProcessName "opsi*"')
```

• processCall(<string>) : string (exitcode) //since 4.11.6.1 [W/L/M]
Starts the command binary <string> as process and returns te exitcode

---

Chapter 9. Syntax and Meaning of Primary Sections of a opsi-script Script [W/L/M]  147 / 281

---

147 / 281
set $exitcode$ = processCall('setup.exe /S')

Is a shortcut for this expression:

Winbatch_setup
set $exitcode$ = getLastExitcode

In fact `processCall` is internal a winbatch call, so all the `winbatch` modifiers are also allowed for `processCall`:

- `/LetThemGo`
  This is the contrary to `/WaitOnClose`. It is used if `opsi-script` shall proceed while the started processes run in their own threads.

- `/TimeOutSeconds <seconds>`
  A timeout setting. After waiting `<seconds>`, `opsi-script` will end the process.
  Since version 4.11.3, `/TimeOutSeconds` may be used without a waiting condition (e.g. `/WaitForProcessEnding`) but not in combination with `/WaitSeconds`.
  Since version 4.11.4.6 the time progress from start until timeout is displayed by the progressbar.

- `/WaitSeconds [number of seconds]`
  If a call includes the parameter `/WaitSeconds [number of seconds]`, then `opsi-script` is waiting for `[number of seconds]` before proceeding. In the default configuration, we also wait for any programs that are currently running to finish. If we combine the parameter `/WaitSeconds` with the option `/LetThemGo`, then `opsi-script` continues processing after the waiting time is finished.

- `/WaitForProcessEnding <program name>`
  Waits for the process called `<program name>` to end.
  Should be combined with `/TimeOutSeconds`.

- `/32Bit //since 4.11.3.5 [W]`
  This is the default. The paths within the section are assumed to be 32 bit paths.
  Example: `c:\windows\system32\regedit.exe` calls (even when running on a 64 bit system) the 32 bit `regedit.exe`.

- `/64Bit //since 4.11.3.5 [W]`
  The paths within the section are assumed to be 64 bit paths.
  Example: `c:\windows\system32\regedit.exe` executes (running on a 64 bit system) the 64 bit `regedit.exe`.

- `/SysNative //since 4.11.3.5 [W]`
  The paths within the section are assigned according to the OS architecture interpreted.
  Example: `c:\windows\system32\regedit.exe` running on a 64bit system calls the 64 bit `regedit.exe` and running on a 32 bit system the 32 bit `regedit.exe`.

9.12. Special Commands [W/L/M]

- **Killtask <process name>` : noresult` [W/L/M]**
  tries to stop all processes that execute the program named by the string expression.
  E.g.

  killtask "winword.exe"

- **ChangeDirectory <directory>` : noresult` //since 4.11.2.6 [W/L/M]**
  Set the given directory as working directory of the `opsi-script`. Affects all subsequent actions (e.g. winbatch sections) and will be reset at the end of a script. Beispiel:

  ChangeDirectory "%SCRIPTPATH%\program"
• **UpdateEnvironment** //since 4.11.5.1 [W]
  Sends Windows the signal to reload the environment values from the registry. This statement may be called after any process that may have changed the environment (e.g. Registry section or setup program). Even if the program that runs after the opsi-script gets the new environment, the next process that is started (via DosBatch or Winbatch) from this opsi-script instance will still inherit the old environment. To start a subsequent process with the new environment you have to use `winbatch` with the `/RunElevated` parameter. Works only with NT6 and up.

Example:

```plaintext
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/M]
  breaks the program execution for <string> seconds. <string> has to represent an Integer Value

• **markTime** [W/L/M]
  sets a time stamp for the current system time and logs it.

• **diffTime** [W/L/M]
  logs the time passed since the last `markTime`.

• **runningInWanMode** //since 4.12.4.17 [W/L/M]
  Returns true if the script is running in the WAN mode. This means that the constant `%opsiserviceurl%` points not to the opsi server but to localhost.

### 9.13. Control the logging

• **comment** <string> or **comment** = <const string> [W/L/M]
  writes the value of the String expression resp. the sequence of characters into the log file.

• **LogError** <string> or **LogError** = <const string> [W/L/M]
  writes additional error messages to the log file and increments the error counter by one.
• **LogWarning** <string> or **LogWarning** = <const string> [W/L/M]
  writes additional warning messages to the log file and increments the warning counter by one.

• **includelog** <file name> <tail size> //since 4.11.2.1 [W/L/M]

• **includelog** <file name> <tail size> [<encoding>] //since 4.11.4.1 [W/L/M]
  Includes the file <file name> as a log file, where the last <tail size> lines of the full log are written into this log file. If you start another program that produces a log file, you could see that other program’s log file in the opsi-script script log using this command.
  Since version 4.11.3.2, a negative <tail size> can be given, which will then include the first <tail size> lines from the top of the log file (referred to as **Head** mode).
  Since version 4.11.4.1 there is an optional third parameter which may be used to give the encoding of the file to include. You may give one of the well known encodings described in the encodings chapter. If you give *auto* opsi-script try to detect the encoding (and may fail). The default is *system* which means the default OS encoding is used.
  Example:

  ```
includelog "%Scriptpath\test-files\10lines.txt" "5"
includelog "%Scriptpath\test-files\10lines_utf16.txt" "5" "ucs2be"
```

**see:** [opsi-script encoding](http://example.com)

• **SetConfidential** <secret string> [W/L/M]
  This is to prevent confidential information (like passwords) from being logged. In the logfile the confidential information will be replaced by (confidential).
  When the loglevel is set to 9, the confidential information will be logged.
  (since version 4.11.3.5)

Example:

```
message "SetConfidential"
SetConfidential "forbidden"
comment "This is a forbidden string"
comment "shown in the 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/M]
  This function should be used to get confidential strings from an other string function without without logging the secret string. The Function work in the following sequence:
  1. Backup the current log level.
  2. Set the log level to Warning. (4)
  3. Resolve the given string expression (for example calling the given string function).
  4. Add the resulting string to the list of confidential strings that should be not logged.
  5. Restore of the initial log level.
  6. Return the resulting string.

Example:
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: should be in the log file: This is a ***(confidential)*** string

see also : [SetConfidential] see also : [GetConfidentialProductProperty] see also : [asConfidential_list]

asConfidential( <secret stringlist expression> ) : stringlist //since 4.12.4.15 [W/L/M]
This function should be used to get confidential stringlist from an other stringlist function without without logging the secret strings.
The Function work like the string function asconfidential, but for stringlists.


• Message <string expression>
bzw.
Message = <sequence of characters>
lets opsi-script display the value of the String expression resp. the sequence of chars in the batch window in the top information line.
The text is kept as long as no new message is set.
Example:
Message "Installation von " + $productid$

• ShowMessageFile <file name>
interprets the String expression as text file name, tries to read the text and show it in a user information window. Execution stops until
the user confirms reading. E.g. by a command like
ShowMessageFile "p:\login\day.msg"

one can realize a "Message of the Day" mechanism.

• ShowBitmap [ <image name> ] [ <inscription> ]
places the image denoted by the <image name> (in BMP, JPEG or PNG format, size 160x160 pixel) and shows the inscription.
<image name> and <inscription> are String expressions.
Example:
ShowBitmap "%scriptpath%" + $ProductId$ + ".png" "$ProductId$"

• Pause <string> or Pause = <const string>
display the text given as a String expression or as a sequence of chars in a information window waiting until the user confirms the continuation.

see also : [SetConfidential] see also : [GetConfidentialProductProperty] see also : [asConfidential_list]
Stop <string> or stop = <const string>
halt program execution if the user confirms it. The String expression resp. the (possibly empty) sequence of chars explain to the user what is supposed to be stopped.

setActionProgress <string>: noresult //since 4.11.3 [W/L/M]
Transfers <string> as ActionProgress of the running script to the opsi server. By Default the ActionProgress is installing while a script is running. The value of ActionProgress is displayed at the configured.

9.15. Commands for userLoginScripts / User Profile Management

GetScriptMode //since 4.11.2.1
give one of the possible values Machine,Login:
- Machine - the script is not running as userLoginScript
- Login - the script is running as userLoginScript

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

GetLoggedinUser //since 4.11.1.2
GetUsercontext //since 4.11.1.2
returns the username in whose context the opsi-script is just running.
see also: [GetUsercontext]

saveVersionToProfile //since 4.11.2.1
save productversion-packageversion to local profile
It is designed to be used in userLoginScripts.
This command is used in combination with readVersionFromProfile or scriptWasExecutedBefore. It marks that the userLoginScript for this product in this product version and package version was executed for the actual user. The information is saved at the file "%CurrentAppdataDir%%opsi.org\userLoginScripts.ini"

readVersionFromProfile //since 4.11.2.1
returns a string with the productversion-packageversion for the running opsi product which was read from local profile. See also: saveVersionToProfile
It is designed to be used in userLoginScripts.

scriptWasExecutedBefore //since 4.11.2.1
This Boolean function scriptWasExecutedBefore checks if there is a version stamp in the profile (like you may do with the readVersionFromProfile command) It returns true if saved and running productversion-packageversion are identical. Then it set a new stamp to the profile (like you may do with the saveVersionToProfile command). So you may just use this single command in a if statement.
It is designed to be used in userLoginScripts.

isLoginScript //since 4.11.2.1
This boolean function returns true if the script is running as userLoginScript. See also: GetScriptMode
see also : [GetScriptMode]

## 9.16. for to do loop

Useful for multiple calls of a single command or of a sub-section

Syntax:

```
for %<temporary string variable>% = <start string> to <end string> do <one statement> //since 4.11.5 [W/L/M]
```

The temporary variabale %<temporary string variable>% must not be declared and is available in the called sub-section as constant.

Example:

Code from opsi-script-test:

```plaintext
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%'
```

produces the 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: ""

--- 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

9.17. Switch / Case Statement [W/L/M]

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

9.18. Conditional Statements (if Statements)

In primary sections, the execution of a statement or a sequence of statements can be made dependent on some condition.

Example
9.18.1. General Syntax

The syntax of the complete if statement is:

```plaintext
if <condition>
  <sequence of statements>
else
  <sequence of statements>
endif
```

The else part may be omitted.

If statements may be nested. That is, in the sequence of statements that depend on an if clause (no matter if inside the if or the else part) another if statement may occur.

<condition> is a <Boolean expression>. A Boolean (or logical) expression can be constructed as a (String) value comparison, by Boolean operators, or by certain function calls which evaluate to true or false. Up to now these Boolean values cannot be explicitly represented in a opsi-script script).

9.18.2. Boolean Expressions

The String comparison (which is a Boolean expression) has the form

```plaintext
<String expression> <comparison sign> <String expression>
```

where <comparison sign> is one of the signs

```plaintext
< <= >=
```

String comparisons in opsi-script are case independent.

Inequality must be expressed by a NOT() expression which is presented below.

There is as well a comparison expression for comparing Strings as (integer) numbers. If any of them cannot be converted to a number an error will be indicated. This number comparison expression has the same form as the String comparison but for an INT prefix of the comparison sign:

```plaintext
<String expression> INT<comparison sign> <String expression>
```

Such, we can build expressions as

```plaintext
if $Name1$ INT<= $Name2$
```

or

```plaintext
if $Number10$ INT>= $Number20$
```

Boolean operators are AND, OR, and NOT() (case does not matter). If b1, b2 and b3 are Boolean expressions the combined expressions

```plaintext
b1 AND b2
b1 OR b2
NOT( b3 )
```

are Boolean expressions as well denoting respectively the conjunction (AND), the disjunction (OR) and the negation (NOT).

A Boolean expression can be enclosed in parentheses (such producing a new Boolean expression with the same value).
The common rules of Boolean operator priority (”and” before ”or”) are at this moment not implemented. An expression with more than one operator is interpreted from left to right. For clarity, in a Boolean expression that combines AND and OR operators parentheses should be employed, e.g. we should explicitly write b1 OR (b2 AND b3) or (b1 OR b2) AND b3

The second example describes what would be executed if there were no parentheses - whereas the common interpretation would run as the other line indicates.

Boolean operators can be conceived as special Boolean valued functions (the negation operator demonstrates this very clearly).

There are some more Boolean functions implemented. Every call of such a function constitutes a Boolean expression as well:

• FileExists (<file name>) : bool [W/L/M]
  returns true if the denoted file or directory exists otherwise false.

• FileExists32(<file name>) see Chapter 64 Bit support

• FileExists64(<file name>) see Chapter 64 Bit support

• FileExistsSysNative(<file name>) see Chapter 64 Bit support

• FileOrFolderExists (<file or folder path> [,<access str>]) : boolean //since 4.12.4.14 [W/L/M]
  returns true if the denoted file or directory exists, otherwise false. + The optional second parameter <access str> is only for Windows: You should enter 32bit, 64bit or sysnative. Otherwise, sysnative is the default for <access str>. see Chapter 64 Bit support
  see also: [DirectoryExists]
  see also: [FileExists]

• DirectoryExists (<path>) : bool //since 4.12.1 [W/L/M]
  Tests if <path> points to a directory.
  <access str> = one of 32bit, 64bit, sysnative; default sysnative; ignored at non windows
  see also: [FileOrFolderExists]
  see also: [FileExists]

Examples:
if ($INST_SystemType$ = "64 Bit System")
    set $ConstTest$ = "true"
    Set $tmp$ = "C:\Windows\system32\Boot"
    set $tmp1$ = "64bit"
    set $CompValue$ = boolToString(DirectoryExists($tmp$, $tmp1$))
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif
    set $ConstTest$ = "true"
    Set $tmp$ = "C:\Windows\system32\Boot"
    set $tmp1$ = "sysnative"
    set $CompValue$ = boolToString(DirectoryExists($tmp$, $tmp1$))
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif
    set $ConstTest$ = "true"
    Set $tmp$ = "C:\Windows\system32\Boot"
    ; fall back to sysnative
    set $CompValue$ = boolToString(DirectoryExists($tmp$))
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif
    set $ConstTest$ = "false"
    Set $tmp$ = "C:\Windows\system32\Boot"
    set $tmp1$ = "32bit"
    set $CompValue$ = boolToString(DirectoryExists($tmp$, $tmp1$))
    if ($ConstTest$ = $CompValue$)
        comment "passed"
    else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
    endif
endif

• fileIsSymlink (<file name>) // since 4.12.4.21 [W/L/M]
  returns true if the denoted file exists and is a symbolic link, otherwise false.

• LineExistsIn (<string>, <file name>) : bool [W/L/M]
  returns true if the text file denoted by <file name> contains a line as specified in the first parameter where each parameter is a String expression. Otherwise (or if the file does not exist) it returns false.

• LineBeginning_ExistsIn (<string>, <file name>) : bool [W/L/M]
  returns true if there is line that begins with <string> in the text file denoted by <file name> (each parameter being a string expression). Otherwise (or if the file does not exist) it returns false.

• LineContaining_ExistsIn(<string>, <file name>) : bool [W/L/M]
  returns true if there is line that contains <string> in the text file denoted by <file name> (each parameter being a string expression). Otherwise (or if the file does not exist) it returns false.

• XMLAddNamespace(<XMLfilename>, <XMLelementname>, <XMLnamespace>)
  inserts a XML namespace definition into the first XML element with the given name (if not existing). It gives back if an insertion took place. (The opsi-script XML patch section need the definitions of namespace.)
  The file must be formatted that an element tag has no line breaks in it. For an example, cf. cookbook Section 12.7, “Inserting a Name
• **XMLRemoveNamespace**(<XMLfilename>, <XMLelementname>, <XMLnamespace>)
removes the XML namespace definition from the XML element. It gives back if an removal took place. We need this to simulate that an
original file is unchanged. For an example, cf. cookbook Section 12.7, “Inserting a Name Space Definition Into a XML File”.

• **HasMinimumSpace**(<Laufwerksname>, <Kapazität>)
returns true if at least a capacity capacity is left on drive drivenname. capacity as well as drivenname syntactically are String expressions.
The capacity may be given as a number without unit specification (then interpreted as bytes) or with unit specifications "kB", "MB", or
“GB” (case independent).
Example:

```
if not (HasMinimumSpace "%SYSTEMDRIVE%", "500 MB")
LogError "Not enough space on %SystemDrive%, 500MB on drive %SystemDrive% needed"
isFatalError
endif
```

• **opsiLicenseManagementEnabled** : bool
returns true if the opsi license management module is enabled.

• **runningAsAdmin**
Returns true if the currently running script was executed with Administrator privileges.
Available since 4.11.1.1

• **isLoginScript**
Returns true if the currently running script was called as userLoginScript using the opsi extension User Profile Management.
Available since 4.11.2.1
see also : [isLoginScript]

• **contains**( <str>, <substr> ) : bool //since 4.11.3: true if <substr> in <str> [W/L/M]
Boolean function which returns true if <str> contains <substr>. This function is case sensitive.
see also : [contains]

• **isNumber**( <str> ) : bool //since 4.11.3: true if <str> represents an integer [W/L/M]
Boolean function which returns true if <str> represents an integer.
see also : [isNumber]

• **runningOnUefi**
Boolean function which returns true if the running OS was booted in UEFI mode.
Available since 4.11.4.3

• **runningInPE** //since 4.12.0.13: [W/L/M]
true if the running OS is a Windows PE

• **runningInWanMode** //since 4.12.4.16: [W/L/M]
Boolean function which returns true if at the running opsi-service context the opsiserver = localhost is.
• `isDriveReady(<drive letter>) //since 4.11.4.4: [W]`
  true: if the drive can be accessed

• `saveTextFile(<list>, <filename>): bool [W/L/M]`
  true: if list is successfully written to file

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

• `saveUnicodeTextFile(<list>, <filename>, <encoding>): bool //since 4.12.4.14 [W/L/M]`
  specific for unicode encoding
  returns true: if list is successfully written to unicode file [W/L/M]

• `CompareDotSeparatedNumbers(<str1>, <relation str>, <str2>): bool //since 4.11.5.2: [W/L/M]`
  compares two strings of the form `<number>[.<number>[.<number>[.<number>]]]`
  by the `<relation str>` which may be one of `<?,=,>=,>`. 
  Since version 4.12.4.28:
  • If while comparing a pair of `<number>` at least one of them has leading zeros, then the compare is done as these `<number>` are the fractional part of a floating number. E.g. `17 > 018` because `0.17 > 0.018`.
  • If while comparing a pair of `<number>` at least one of them has as last char not a not a numeric char then the compare uses in the first step the leading number part. In case of equality then a compare of the trailing chars are done. E.g. `1.23a < 1.23b` and `1.24a > 1.23b`. see also: string function `CompareDotSeparatedNumbers(<string1>, <string2>): [CompareDotSeparatedNumbers_str]` see also: string function `CompareDotSeparatedStrings(<string1>, <string2>): [CompareDotSeparatedStrings_str]` see also: `CompareDotSeparatedStrings_bool`

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($string1$, "=", $string2$)
    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 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
Additional since 4.12.4.28:
if CompareDotSeparatedNumbers("4.2.2", ">", "4.2.00079")
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif
if CompareDotSeparatedNumbers("0.9.8h", ">", "0.9.8e")
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif

* CompareDotSeparatedStrings(<str1>,<relation str>,<str2>) : bool //since 4.11.5.2: [W/L/M]
  compares two strings of the form <str>[.<str>[.<str>[.<str>]]]
  by the <relation str> which may be one of [<,<=,=,>,>=].
  see also: string function CompareDotSeparatedStrings(<string1>, <string2>) : [CompareDotSeparatedStrings_str]
  see also: [CompareDotSeparatedNumbers_bool]
  see also: [CompareDotSeparatedNumbers_str]
• `boolToString(<boolean expression> ) : bool string (true/false) // since 4.12.0.0 [W/L/M]`

• `stringToBool(<string expression: true/false> ) : boolean // since 4.12.0.0 [W/L/M]`

• `RegKeyExists(<regkey>) : bool //since 4.12.0.16 [W]
  Check if the given string expression <regkey> exists as registry key. If the registry key was found the result value ist true in all other cases false.
  By Default the registry access mode is sysnative. Using the optional second parameter <access str>, the access mode can be explicitly given. In this case it has to be one of the following values: 32bit, sysnative, 64bit.
  (see also: Chapter 64 Bit)

Examples:

```
RegKeyExists("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon")
RegKeyExists("HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general","32bit")
```

• `RegVarExists(<regkey>, <var str> ) : bool //since 4.12.0.16 [W]
  Check if the given string expression <regkey> exists as registry key and if there is a variable with name <var str>. If both was found the result value ist true in all other cases false.
  By Default the registry access mode is sysnative. Using the optional second parameter <access str>, the access mode can be explicitly given. In this case it has to be one of the following values: 32bit, sysnative, 64bit.
  (see also: Chapter 64 Bit)

Examples:

```
RegVarExists("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon","Shell")
RegVarExists("HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general","bootmode","32bit")
```

• `runningWithGui : bool` //since 4.12.3.6 [W/L/M] true: if the running OS has a GUI (at Win+Mac always true)[M/L/W]

• `isPingReachable(<host>) : boolean //since 4.12.3.6 [W/L/M]
  Gives back true if the host given by <host> is reachable by ping. <host> may be a IP-Number or a IP-Name.

## 9.19. Include Commands

Using include commands can lead to confusing code.
Use with caution if you are a beginner.

### 9.19.1. Include Commands: Syntax

The `include_*` commands may be used to add external script files to the currently-running script at runtime. The `include_*` command can include external script files either as an insert (after the current line) or as an append (after the last line). The include commands may be used anywhere in a primary section. The external script files may contain their own include commands.

The include commands are available since version 4.11.3
include_insert <file name>

inserts <file name> after the current line into the running script. So the first line of the included file is the next line that will be executed by opsi-script.

include_append <file name>

appends the content of <file name> to the running script. This kind of insert is normally used to include sections from a library.

In both cases <file name> is:

- A complete path to an existing file. [W/L/M]
- A existing file in %ScriptPath% [W/L/M]
- A file in %opsiScriptHelperPath%/lib [W]
  Is equivalent to: %ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib
- A existing file in %ScriptPath%/../lib [W/L/M]
- A existing file in %opsiScriptDir%/lib [W]

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.

Example:
When we run that contains the following commands:

```
[Actions]
include_append "section_Files_del_tmp_dummy.opsiinc"
include_insert "include-test1.opsiinc"
```

The file include-test1.opsiinc is run first. The contents of the included file include-test1.opsiinc are:

```
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
```

The contents of the included file section_Files_copy_inctest.opsiinc are:

```
[Files_copy_inctest]
copy "%scriptpath%\test-files\dummy.txt" "c:\opsi.org\tmp"
```

Since the call to Files_del_tmp_dummy happens inside of include-test1.opsiinc without section_Files_del_tmp_dummy.opsiinc being referenced inside of include-test1.opsiinc, we must call include_append "section_Files_del_tmp_dummy.opsiinc" at the very beginning of our script. Otherwise, opsi-script will report that Files_del_tmp_dummy is not defined.

The contents of the included file section_Files_del_tmp_dummy.opsiinc are:

```
[Files_del_tmp_dummy]
del -f "c:\opsi.org\tmp\dummyt.txt"
```
9.19.2. Include Commands: Library

The following include files are shipped with version 4.11.3, and are located in %OpsiScriptDir%\lib:

**insert_check_exit_code.opsiinc:**

```opsi
; ospi include file
DefVar $ExitCode$
include_append "section_sub_check_exitcode.opsiinc"
```

**insert_get_licensekey.opsiinc:**

```opsi
; ospi include file
DefVar $LicenseRequired$
DefVar $LicenseKey$
DefVar $LicensePool$
include_append "section_sub_get_licensekey.opsiinc"
```

**section_sub_check_exit_code.opsiinc:**

```opsi
; ospi include file
[Sub_check_exitcode]
comment "Test for installation success via exit code"
set $ExitCode$ = getLastExitCode
; informations to exit codes see
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"
            exitWindows /Reboot
        else
            if ($ExitCode$ = "3010")
                comment "ERROR_SUCCESS_REBOOT_REQUIRED 3010 A restart is required to complete the install. This message is indicative of a"
                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:**
9.20. Subprogram Calls

Statements in primary sections which refer to instructions declared elsewhere are subprogram calls.

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

In this example the statement:

```
sub_install_winXP
```

"calls" the section titled \[sub_install_winXP\] which is placed somewhere else in the script. E.g. we may have

```
[sub_install_winXP]
Files_copy_XP
WinBatch_SetupXP
```

Generally, there are three ways to place the referred instructions:

- The most common target of a sub program call is some other internal section in the very script file where the calling statement is placed (as in the example).
- We may put the referred instructions into another file which serves as an external section.
- Any String list can be used as list of instructions for a sub program call.

We describe the syntax of sub program calls in detail:
9.20.1. Syntax of Procedure Calling

Formally, the syntax can be given by:

\[ \langle \text{proc. type} \rangle \langle \text{proc. name} \rangle | \langle \text{External proc. file} \rangle | \langle \text{String list function} \rangle \]

This expression may be supplemented by one or more parameters (procedure type dependent).

That means: A procedure call consists of three main parts.

The first part is the subprogram type specifier. Examples of type names are `Sub` (we call a procedure of type sub that is a again a primary section) or `Files` and `WinBatch` (calls of special secondary sections). The complete overview of the existing subprogram types is given at Subprogram Calls.

The second part determines where and how the lines of subprogram are to be found.

1. The subprogram is a sequence of lines situated in the executed `opsi-script` script as another internal section. Then a name (constituted from letters, digits, and some special characters) has to be appended to the type specifier (without space) in order to form an unique section name.

   sub_install_winXP
   or
   files_copy_winXP

   Section names are case independent as any other string.

2. If the type specifier stands alone a String list expression or a String expression is expected. If the expression following the type specifier cannot be resolved as a String list expression (cf. case (3)) it is assumed to be a String expression. The string is then interpreted as a file name. `opsi-script` tries to open the file as a text file and interprets its lines as an external section of the specified type.

   E.g.
   sub "p:\install\opsiutils\mainroutine.ins"

   tries to execute the lines of mainroutine.ins as statements of a sub section.

The search rule is:

<file name> may be:

- A complete path to an existing file. [W/L/M]
- A existing file in %ScriptPath% [W/L/M]
- A file in %opsiScriptHelperPath%\lib [W]
  Is equivalent to: %ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib
- A existing file in %ScriptPath%/../lib [W/L/M]
- A existing file in %opsiScriptDir%\lib [W]

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.

1. If the expression following a pure section type specifier is resolvable as a String list expression the lines of the list are interpreted as the statements of the section.

   This mechanism can e.g. be used to load a file that has unicode format and then treat it by the usual mechanisms

   ```
   registry loadUnicodeTextFile("\scriptpath\\opsiorg\key.reg") /regedit
   ```

   Syntactically, this line is composed of three main parts:
   * `registry`, the core statement specifying the section type,
   * `loadUnicodeTextFile(...)`, a String list expression specifying how to get the lines of a registry section resp. its surrogate.
   * `/regedit`, parametrizing the registry call.

   In this example, the call parameter already gives an example for the third part of a subsection call:

   The third part of a procedure call comprises type specific call options.

   For a reference of the call options cf. the descriptions of the section calls in Secondary Sections.
9.21. Controlling Reboot

The command `ExitWindows` is used to control reboots, shutdown and similar actions which should take place after the `opsi-script` itself is terminated. The name of the command and the fact that there is no `ExitWindows` without modifier has historical reasons: Working on Windows 3.1 you could exit windows to go back to the DOS level.

- **ExitWindows /RebootWanted**
  
  DEPRECATED: a reboot request is registered which should be executed when all installations requests are treated, and the last script has finished.

  In fact, this command is now treated as an `ExitWindows /Reboot` (since otherwise an installation could fail because a required product is not yet completely installed).

- **ExitWindows /Reboot**

  triggers the reboot after `opsi-script` has finished the currently treated script.

- **ExitWindows /ImmediateReboot**

  breaks the normal execution of a script anywhere inside it. When this command is called `opsi-script` runs as directly as possible to its end entailing the system `ExitWindows` call. In the context of an installed `opsi-client-agent` it is guaranteed that after rebooting `opsi-script` runs again into the script that was aborted. Therefore, the script has to take provisions that the execution continues after the point where it was left the turn before (otherwise we may get an infinite loop …) Cf. the example in this section.

- **ExitWindows /ImmediateLogout**

  The normal execution of a script breaks at the point of the call, and the `opsi-script` stops running. This behaviour is needed if an automated user log in for some other user shall take place (cf. Section 12.3, “Script for installations in the context of a local user”).

- **ExitWindows /ShutdownWanted**

  sets a flag in the registry that the PC shuts down when all installations requests are treated, and the last script has finished.

How flags may be set to ensure that the script does not run into an infinite loop when `ExitWindows /ImmediateReboot` is called we demonstrate by the following code fragment:
DefVar $OS$
DefVar $Flag$
DefVar $WinstRegKey$
set $OS$=EnvVar("OS")
if $OS$="Windows_NT"
Set $WinstRegKey$ = "HKLM\SOFTWARE\opsi.org\winst"
Set $Flag$ = GetRegistryStringValue("\["+$WinstRegKey$+"\] RebootFlag")
if not ($Flag$ = "1")
;========================
; Statements BEFORE Reboot
Files_doSomething
; initialize reboot ...
Set $Flag$ = "1"
Registry_SaveRebootFlag
ExitWindows /ImmediateReboot
else
;========================
; Statements AFTER Reboot
; set back reboot flag
Set $Flag$ = "0"
Registry_SaveRebootFlag
; the work part after reboot:
Files_doMore
endif
endif

[Registry_SaveRebootFlag]
openKey [[$WinstRegKey$]]
set "RebootFlag" = "$Flag$"

[Files_doSomething]
; a section executed before reboot

[Files_doMore]
; a section executed after reboot

### 9.22. Abort script and keep track of failed installations

If a product installation fails, then this should be signaled to the server.

Due to the fact that there is no automatic method that detects a failed installation, testing for a failed installation has to be done using script commands.

To indicate in an *opsi-script* script that the installation is failed we have to call the statement:

**isFatalError**

If this statement is called, then *opsi-script* stops the normal execution of the script and sets the product result to *failed* (otherwise it is *success*).

Since 4.11.3.2 there is a new variant of this command:

- **isFatalError** <string>

  in this case, a short error message string is passed as *actionProgress* to the opsi-server and displayed in the opsi-configed.

  For example, a “fatal error” shall be triggered if there is not as much space left as it is needed for an installation:
It is also possible to state `isFatalError` depending on the number of errors which occurred in some critical part of an installation script. In order to do this we initialize the error counting by the command

* `markErrorNumber`  
  Initialize the error counting.

The number of execution errors which occur after setting the counter can be queried by the number valued function `errorsOccurredSinceMark`.

* `errorsOccurredSinceMark`  
  We can evaluate the result in a numerical comparison condition (that as yet is only implemented for this expression). E.g. we may state

  ```plaintext
  if errorsOccurredSinceMark > 0
  ```

For increasing the number of counted errors depending on certain circumstances (that do not directly produce an error) we may use the `logError` statement.

We may test this by the following script example:

```plaintext
markErrorNumber
; Errors occurring after this mark are counted and
; will possibly be regarded as fatal
logError "test error"
; we write "test error" into the log file
; and increase the number of errors by 1
; for testing, comment out this line

if errorsOccurredSinceMark > 0
; we finish script execution as quick as possible
; and set the product state to "failed"
  isFatalError
  ; but comment writing is not stopped
  comment "error occurred"
else
; no error occurred, lets log this:
  comment "no error occurred"
endif
```

* `isSuccess`  
  Abort the script as successful.
• noUpdateScript [since 4.11.3.7] [W/L/M]
  Do not run an update script after setup even if there is one.

• isSuspended [since 4.11.4.1] [W/L/M]
  Abort the script without notice to the server. The action request remain unchanged.

9.23. Local functions [W/L/M]

Since version 4.12, the opsi-script has also local functions.

An example:

```opsi
DefFunc myFunc(val $str1$ : string, $str2$ : string) : string
  set $result$ = $str1$ + $str2$
endfunc
```

9.23.1. Concept

There are a lot possibilities to structure opsi-script code:

• sub Sections
• sub Sections in external files
• include Statements

But all these possibilities are not functional to create reusable external code that can be exchanged between scripts or opsi administrators without problems. The reason is, that this code is not encapsulated and use global variables. The defined local functions presented here now solves this problem. With this concept it is possible to write functions that can be collected and maintained in external libraries.

In consequence we will start to build up a central opsi-script library which is maintained by uib and the opsi community.

In order to reach this target we have implemented the following concepts:

• Functions with return value:
  The functions have a return value which is of the type string or stringlist. Executing such function can be performed wherever a string expression or a stringlist is expected.
  Functions with no return value are also allowed and have to be declared as void (since 4.12.0.16).

• Freely definable function call parameters:
  Parameters can be passed to a function. These parameters are defined when the function is actually declared. The call parameters can be of type string or stringlist.
  The call parameters are available as local variables within the function. The call parameters can be passed as CallByValue or callByReference. CallByValue is the default. That means, if no call method is specified explicitly, then CallByValue will be applied. In the case that CallByValue needs to be explicitly specified, then the keyword val should be used. CallByValue means, that the value of a variable used during the call is copied to the call variable. CallByReference must be specified explicitly using the keyword ref. callByReference means that a connection is created between the variable used as parameter when calling the function and the local variable that represents the call parameter inside the function. Changing the local variable of the call parameter has a direct effect on the variable used during such call.

• Local Variables:
  A function contains local variables: Implicitly, the call parameters are available as local variables and the variable $result$ which is from the type of the returned value. Further variables can be defined within the function.
  All these variables are local, which means that they are only visible within this function. A local variable with the same name of a global variable masks the corresponding global variable within the function.

• Nested functions:
  A local function can in turn have one or even more definitions of local functions. These functions are only visible within the function in which they are defined.

• Recursive calls:
  A function can call itself recursively.
• Primary and secondary sections within functions:
The function body can contain its own sections of it. These are local to this function, that means that these sections are only visible within the function.

9.23.2. Syntax

Definition

\[
\text{DefFunc } \text{<func name>}(\text{[calltype parameter ptype]}[,\text{[calltype parameter ptype]}]) : \text{ftype}
\text{<function body>}
\text{endfunc}
\]

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 \[\text{val} | \text{ref}\]. \text{val}=Call by Value, \text{ref}=Call by Reference. Default: \text{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 wether \text{string} or \text{stringlist}.
- **ftype** is the type of data of the function wether \text{string}, \text{stringlist} or \text{void}. \text{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.

A local function has to be defined **before** you can call the function.

9.23.3. Examples

Simple function that connects two strings:

```
[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",$str1$)
```

Expected results:

- heho
- he42921.809
- heha

Function of the type \text{stringlist} which will deliver a \text{string} and a \text{stringlist}:
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", $list2$)

Expected results:

- $list1$ = [hi,opsi]

Function of type string to which a boolean string will be deliver:

DefFunc myFunc2($str1$ : string) : string
  set $result$ = booltostring($str1$)
endfunc

if stringtobool(myfunc2('1 > 0'))
  comment "true"
else
  comment "false"
endif

Expected results:

- true

Function of the type string to which a string is passed with local variable:

DefVar $mystr$

DefFunc myFunc3($str1$ : string) : string
  DefVar $locstr1$
  set $locstr1$ = '123'
  set $result$ = $locstr1$ + $str1$
endfunc

set $mystr$ = myFunc3("he")

Expected results:

- 123he

Function of the type string to which a string is passed with local variable and nested function:

DefVar $mystr$

DefFunc myFunc4($str1$ : string) : string
  DefVar $locstr1$
  DefFunc myFunc5($str1$ : string) : string
    set $result5$ = 'inner' + $str1$
  endfunc
  set $locstr1$ = '123'
  set $result$ = $locstr1$ + $str1$
endfunc

set $mystr$ = myFunc4("outer")

Expected results:
Simple function of type `string` which pass a `string` by reference with a local variable:

```plaintext
(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 $mystr$ = $str1$ + $str2$
```

Expected results:
- `123setinlocal`
- `hasetinlocal`

Function of type `stringlist` which will pass a variable of type `stringlist` with a `call by reference` also with a local `stringlist` variable:

```plaintext
(actions)
DefVar $mystr$
DefStringlist $list1$
DefStringlist $list2$
set $list2$ = 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,$list2$)
```

Expected results:
- `$list1$ = [a,b,c,setinlocal]`
- `setinlocal`

Function of type `stringlist` which pass a `string` with a local variable and a local secondary section:

```plaintext
(actions)
DefStringlist $list1$
DefFunc myFunc8($str1$ : string) : stringlist
DefStringlist $loclist1$
set $loclist1$ = getoutstreamfromsection("shellInAnIcon_test")
set $result$ = $loclist1$
[shellinanicon_test]
set = x
$str1$
endfunc
set $list1$ = myFunc8('pwd')
```

Expected results:
- `$list1$ = [+ pwd, /home/uib/gitwork/lazarus/opsi-script]`
Function of type **void** (no return value) which pass a **string** with a local variable:

```plaintext
[actions]
ScriptErrorMessages = false
DefVar $str1$

set $str1$ = 'haha'

DefFunc myNoResultFunc(ref $str1$ : string) : void

set $str1$ = "huhu"

endfunc

myNoResultFunc($str1$)

comment "$str1$ is: "+$str1$
```

Expected results:

- $str1$ is: huhu

Function of type **string** with no parameter:

```plaintext
[actions]
ScriptErrorMessages = false
DefVar $str1$

DefFunc myNoParamFunc() : string

set $result$ = "huhu"

endfunc

set $str1$ = myNoParamFunc()

Expected results:

- $str1$ is: huhu
```

### 9.24. Import of libraries of functions [W/L/M]

```plaintext
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/M]
- A existing file in %ScriptPath% [W/L/M]
- A file in %opsiScriptHelperPath\lib [W]
  Is equivalent to: %ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib
- A existing file in %ScriptPath%/../lib [W/L/M]
- A existing file in %opsiScriptDir%\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.
Chapter 10. Secondary Sections

The secondary sections can be called from any primary section but have a different syntax. The syntax is derived from the functional requirements and library conditions and conventions for the specific purposes. Therefore from a secondary section, no further section can be called.

Secondary sections are specific each for a certain functional area. This refers to the object of the functionality, e.g. file system in general, the Windows registry, or XML files. But it refers even more to the apparatus that is internally applied. This may be demonstrated by the the variants of the batch sections (which call external programs or scripts).

The functional context is mirrored in the specific syntax of the particular section type.

10.1. Calling secondary sections

In most cases a secondary section is called by calling the section header as a statement. As a statement means here: no return value is expected.

Example:

```plaintext
shellInAnIcon_say_hello
[shellInAnIcon_say_hello]
echo "Hello World"
```

In the case that you have to call the section with modifiers you have to add these modifiers behind the statement as text (no variables are allowed here):

```plaintext
shellInAnIcon_say_hello /timeoutseconds 20
[shellInAnIcon_say_hello]
echo "Hello World"
```

If you want to examine the output or return values of a section you may use one of the following functions:

- `getOutStreamFromSection(<section with params>)` for `DosInAnIcon (ShellInAnIcon),ExecWith and ExecPython` calls. See also `[getOutStreamFromSection]`
- `getReturnListFromSection(<section with params>)` for `XMLPatch` sections and `opsiServiceCall` sections. See also `[getReturnListFromSection]`

```plaintext
set $list$ = getOutStreamFromSection("shellInAnIcon_say_hello /timeoutseconds 20")
[shellInAnIcon_say_hello]
echo "Hello World"
```

Using this functions, the complete call of the section (inclusive all parameters and modifiers) has to be one string expression. This gives you the possibility to use variables and functions as part of the section call.

If you want to use this advantages without catching the out streams you may use the following statement:

```plaintext
executeSection(<string expr with section call>) //since 4.12.3.9 [W/L/M]
```

within this statement you may call the following section types:

- `winbatch`
- `registry`
- `DosBatch, DosInAnIcon, ShellBatch, ShellInAnIcon`
- `ExecWith, ExecPython`
10.2. Files Sections

A Files section mainly offers functions which correspond to copy commands of the underlying operating system. The surplus value when using the `opsi-script` commands is the detailed logging and checking of all operations when necessary. If wanted overwriting of files can be forbidden if newer versions of a file (e.g. a newer dll-file) are already installed on the system.

10.2.1. Example

A simple Files section could be:

```
[Files_do_some_copying]
copy -sV "p:\install\instnsc\netscape\*.*" "C:\netscape"
copy -sV "p:\install\instnsc\windows\*.*" "%SYSTEMROOT%"
```

These commands cause that all files of the directory `p:\install\instnsc\netscape` are copied to the directory `C:\netscape`, and then all files from `p:\install\instnsc\windows` to the windows system directory (its value is automatically inserted into the constant name `%SYSTEMROOT`). Option `-s` means that all subdirectories are copied as well, `-V` activates the version control for library files.

10.2.2. Modifier

In most cases a Files section will be called without parameters.

There are only some special uses of Files sections where the target of copy actions is set or changed in a certain specified way. We have got the two optional parameters

- `/AllUserProfiles` //since 4.12.4.27 [W/L/M]
- `/AllNTUserProfiles` (discouraged)
- `/AllNTUserSendTo` [W]

Both variants mean:
The called Files section is executed once for each local user. Every copy command in the section is associated with an user specific target directory.

In case other we need to build other user specific path names we can use the automatically set variable `%UserProfileDir%` or since `opsi-script` version 4.11.2 `%CurrentProfileDir%`. With option `/AllUserProfiles` (resp. `/AllNTUserProfiles`) the user specific target directory for copy actions is the user profile directory (that is usually denoted by the user name and is by default situated as a subdirectory of the userappdata directory. In case of option `/AllNTUserSendTo` the target directory is the path of the user specific SendTo folder (for links of the `windows explorer` context menu).

The exact rule for determining the target path for a copy command has three parts:

1. If only the source of a copy action is specified the files are copied directly into the user target directory. We have syntax
   ```
copy <source file(s)>
   ```
   It be equivalent as
   ```
copy <source file(s)> "%UserProfileDir%"
   ```
   or since 4.11.2
   ```
copy <source file(s)> "%CurrentProfileDir%"
   ```
2. If some targetdir is specified and targetdir is a relative path description (starting neither with a drive name nor a backslash) then targetdir is regard as the name of a subdirectory of the user specific directory. I.e.
   ```
copy <source file(s)> <targetdir>
   ```
   is interpreted like:
   ```
copy <source file(s)> "%UserProfileDir%\targetdir"
   ```
   or since 4.11.2
   ```
copy <source file(s)> "%CurrentProfileDir%\targetdir"
   ```
The use of \%CurrentProfileDir\% has the advantage that you may the same Files section with /AllUserProfiles if it is not running as userLoginScript (in Machine script mode) and without /AllUserProfiles if it is running as userLoginScript (in Login script mode).

1. If targetdir is an absolute path it is used as the static target path of the copy action.

There are also the Options:

- /32Bit (Default)
- /64Bit
- /SysNative

which manipulate the file redirection on 64 Bit systems. For more details see Chapter 11, 64 Bit Support on Windows [W]

### 10.2.3. Commands

In a Files section the following commands are defined:

- Copy [W/L/M]
- Delete/Del [W/L/M]
- SourcePath
- CheckTargetPath [W/L/M]
- chmod [L/M]
- hardlink [W/L/M]
- symlink [W/L/M]
- rename [W/L/M]
- move [W/L/M]
- zipfile [W/L/M]
- unzipfile [W/L/M]

Copy and Delete roughly correspond the the Windows shell commands xcopy resp. del.

SourcePath and CheckTargetPath set origin and destination of the forthcoming copy actions (as if we would open two explorer windows for copy actions between them). If the target path does not exist it will be created.

The syntax definitions are:

- **Copy** [-svdunxwnrh] <source(mask)> <target path>

  The source files can be denoted explicitly, using the wild card sign ("*" ) or by a directory name.

  The <target path> is always understood as a directory name. Renaming by copying is not possible. If the target path does not exist it will be created (if needed a hierarchy of directories).

The optional options of the Copy command mean (the ordering is insignificant):

- **s** - We recursive into subdirectories, [W/L/M]
- **e** - Empty Subdirectories.
  If there are empty subdirectories in the source path they will be created in the target directory as well.
- **V** - Version checking [W]
  A newer version of a windows library file is not overwritten by an older one (according primarily to the internal version counting of the file). If there are any doubts regarding the priority of the files a warning is added to the log file.
- v ~ (do not use)
  With Version checking: [W]
  Deprecated: Don’t use it on Systems higher than win2k. Because it checks not only against the target directory but also against
  %System%. use -V instead.

- d ~ With date check: [W]
  A newer .exe file is not overwritten by an older one.

- u ~ We are only updating files: [W]
  A file is not copied if there is a newer or equally old file of the same name.

- x ~ x-tract [W]
  If a file is a zip archive it will be unpacked (Xtracted) on copying.
  Caution: Zip archives are not characterized by its name but by an internal definition. E.g. a java jar file is a zip file. If it is unpacked
  the application call will not work.

- w ~ weak [W]
  We respect any write protection of a file such proceeding "weakly" (in opposite to the default behaviour which is to try to use
  administrator privileges and overwrite a write protected file).

- n ~ no over write [W]
  Existing files are not overwritten.

- c ~ continue [W]
  If a system file is in use, then it can be overwritten only after a reboot. The opsí-script default behaviour is therefore that a file in
  use will be marked for overwriting after the next reboot, AND the opsí-script reboot flag is set. Setting the copy option ~c turns the
  automatic reboot off. Instead normal processing continues, the copying will be completed only when a reboot is otherwise
  triggered.

- r ~ read-only Attribute [W]
  If a copied file has a read-only attribute it is set again (in opposite to the default behaviour which is to eliminate read-only
  attributes).

- h ~ follow symlinks [L] //since 4.11.6.14
  At Linux symlinks to files or directories will be resolved before copy. So not the symlink but its target will be copied.

- Delete [-sfdnrc] <path>

  or

  * Delete [-sfdnrc] <source(mask)>

  deletes files and directories.

Possible options are (with arbitrary ordering)

- s ~ subdirectories
  We recurse into subdirectories. Everything that matches the path name or the source mask is deleted.

  The command
  
  delete -s c:\opsi
  
  Do not mean: remove the directory c:\opsi recursive, but it means: delete starting frm c:\ all occurences of opsí.
  
  This may lead to a complete hard disk scan.

  If you want to delete the directory c:\opsi recursive use the command:
  
  delete -s c:\opsi

  by using a trailing backslash you define that opsí is a directory.

  It is safer to use the command del instead.

- f ~ force
  forces to delete read only files

- r ~ del on reboot [W] since 4.12.4.3 If a file is in use, it can not be deleted at Windows. With the option r in this case the file will be
  marked for deletion at the next reboot. If this is done then opsí-script default behaviour is also that the opsí-script reboot flag is set. This
  leads to a reboot after the script is finished. see also option: c
• c ~ continue with out reboot [W]+ If a file is in use, it can not be deleted at Windows. With the option r in this case the file will be marked for deletion at the next reboot. If this is done then opsi-script default behaviour is also that the opsi-script reboot flag is set. Setting the copy option -c turns the automatic reboot off. Instead normal processing continues, the deleting will be completed only when a reboot is otherwise triggered. Option c make no sense without r and will be ignored.

• d [n] ~ date
  Only files of age n days or older are deleted. n defaults to 1.
  - del [Options] <path[mask]] //since 4.11.2.1
    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

• SourcePath = < source directory>
  Sets <source directory> as default directory for the following Copy and (!) Delete commands.

• CheckTargetPath = <target directory>
  Sets <target directory> as default directory for Copy command. If the specified path does not exist it will be created.

• chmod <mode> <path> //since 4.11.4.1 [L]
  Sets the access rights for <path> to <mode>. <mode> is the numerical (octal) representation (e.g. “755”).

• hardlink <existing file> <new file> // since 4.11.5 [W/L/M]
  A existing <new file> will be over written.
  hardlink works only on filesystems that support hard links like NTFS and standard Linux filesystems.

• symlink <existing file> <new file> // since 4.11.5 [W/L/M]
  A existing <new file> will be over written.
  At Windows is symlink only available since NT6 and up !

• rename <old filename> <new filename> // since 4.11.5 [W/L/M]
  move <old filename> <new filename> // since 4.11.5 [W/L/M]
  There is no difference between rename and move, that are just two names for the same function
  A existing <new file> will be over written.
  Since 4.12.4.31 it is also possible to move or rename directories.

Windows: <new filename> may be located in a differen directory or volume / disk. In the second case (different volume / disk) the file will be copied and than the original file will be deleted.
If it is not possible to create the target file becaus the file is in use, then it can be overwritten only after a reboot. The opsi-script default behaviour is therefore that a file in use will be marked for overwriting after the next reboot, AND the opsi-script reboot flag is set. Setting the copy option -c turns the automatic reboot off. Instead normal processing continues, the copying will be completed only when a reboot is otherwise triggered.
Creating Junctions at Windows is not supported right now.

Linux: <new filename> may be located in a different directory but not in a different filesystem. The Option -c will be ignored at Linux.

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"
movew "$HomeTestFiles\files\temp\move2.txt" "$HomeTestFiles\files\temp\move.txt"

zipfile <source dir> <zip file> // since 4.12.1 [W/L/M]
unzipfile <zip file> <target dir> // since 4.12.1 [W/L/M]

Example:
10.3. Patches-Sections [W/L/M]

A Patches section modifies a property file in ini file format. I. e. a file that consists of sections which are a sequence of entries constructed as settings `<variable> = <value>`. where sections are characterized by headings which are bracketed names like `[sectionname]`.

10.3.1. Example

```
$HomeTestFiles$"+\"dummy.ini\"

[secdummy]
dummy1=add1
add dummy2=add2
add dummy3=add3
add dummy4=add4
add dummy5=add5
add dummy6=add6
set dummy2=set1
addnew dummy1=addnew1
change dummy3=change1
del dummy4
replace dummy6=add6 replace1=replace1
```

produces the following 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
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
replaceEntry dummy6=add6 replace1=replace1
placed in line 7
C:\tmp\testFiles\dummy.ini saved back
```

For more examples, please check the opsi standard product opsi-script-test and in this product the part `$Flag_winst_patches$ = "on"`

Call Parameter

The name of the file to be patched is passed as a parameter.

There are optional modifiers:

- `\AllUserProfiles` (old synonym: `\AllNTUserProfiles`)

  If a patch section is called with this modifier, then all directories under `%UserProfileDir%` will be patched, which means that this patch
is performed for all user profiles. When a Patches is called within a [ProfileActions] section, then the modifier /AllUserProfiles is implicit. In logscript mode, %UserProfileDir% will be interpreted as %CurrentProfileDir%.

(Since Version 4.11.3.2)

• /encoding <encoding> //since 4.12.4.17 [W/L/M]
You can add an encoding parameter for the Patches section. This is needed if the file that you want to modify is not in system encoding.
Example:

Patches_my_win_ini "C:/my_file.ini" /encoding "utf16le"

For allowed encodings see opsi-script encoding

10.3.2. Commands

For a Patches section, we have commands:

• add
• set
• addnew
• change
• del
• delsec
• replace

Each command refers to some section of the file which is to be patched. The name of this section is specified in brackets [] (which do here not mean "syntactically optional"!).

In detail:

• add [section name] <variable1> = <value1>
  This command adds an entry of kind <variable1> = <value1> to section <section name> if there is yet no entry for <variable1> in this section. Otherwise nothing is written. If the section does not exist it will be created.

• set [section name] <variable1> = <value1>
  If there is no entry for <variable1> in section <section name> the setting <variable1> = <value1> is added. Otherwise, the first entry <variable1> = <valueX> is changed to <variable1> = <value1>.

• addnew [section name] <variable1> = <value1>
  No matter if there is an entry for <variable1> in section <section name> the setting <variable1> = <value1> is added.

• change [section name] <variable1> = <value1>
  Only if there is any entry for <variable1> in section <section name> it is changed to <variable1> = <value1>.

• del [section name] <variable1> = <value1>
  resp.
  del [section name] <variable1> removes all entries <variable1> = <value1> resp. all entries for <variable1> in section <section name>.

• delsec [section name]
  removes the section <section name>.

• replace <variable1>=<value1> <variable2>=<value2>
  means that <variable1> = <value1> will be replaced by <variable2> = <value2> in all sections of the ini file. There must be no spaces in the value or around the equal signs.
10.4. PatchHosts Sections [W/L/M]

By virtue of a PatchHosts section we are able to modify a hosts file which is to understand as any file with lines having format

```
IPadress hostname aliases # comment
```

Aliases and comment (and the comment separator #) are optional. A line may also be a comment line starting with #.

The file which is to be modified can be given as parameter of a PatchHosts call. If there is no parameter a file named HOSTS is searched in the directories c:\nfs, c:\windows and %systemroot%\system32\drivers\etc. If no such file is found the PatchHosts call terminates with an error.

In a PatchHosts section there are defined commands:

- `setAddr`
- `setName`
- `setAlias`
- `delAlias`
- `delHost`
- `setComment`

Example:

```
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
```

produces the following 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
```

For more examples, please check the opsi standard product `opsi-script-test` and in this product the part `$Flag_winst_patch_hosts$ = "on"`.

In detail:

- `setaddr <hostname> <ipaddress>`
  sets the IP address for host `<hostname>` to `<IPaddress>`. If there is no entry for host name as yet it will be created.

- `setname <ipaddress> <hostname>`
  sets the host name for the given IP address. If there is no entry for the IP address as yet it will be created.

- `setalias <hostname> <alias>`
  adds an alias for the host named `<hostname>`.

- `setalias <IPadress> <alias>`
  adds an alias name for the host with IP address `<IPadress>`.

- `delalias <hostname> <alias>`
  removes the alias name `<alias>` for the host named `<hostname>`.

- `delalias <IPadress> <alias>`
  removes the alias name `<alias>` for the host with IP address `<IPadress>`.
delhost <hostname> removes the complete entry for the host with name <hostname>.

delhost <IPaddress>
removes the complete entry for the host with IP address <IPadress>.

setComment <ident> <comment>
writes <comment> after the comment sign for the host with host name, IP address or alias name <ident>.

10.5. IdapiConfig Sections

A IdapiConfig section were designed to write parameters in idapi*.cfg files which are used by the Borland Database Engine.

This section type is not supported any more.

10.6. PatchTextFile Sections [W/L/M]

A PatchTextFile section offers a variety of options to patch arbitrary configuration files which are given as common text files (i.e. they can be treated line by line).

An essential tool for working on text files is the check if a specific line is contained in a given file. For this purpose we have got the Boolean functions Line_ExistsIn and LineBeginning_ExistsIn (cf. Boolean Expressions).

10.6.1. Parameter

The text file which is to be treated is given as parameter.

There are optional modifiers:

* /AllUserProfiles (old synonym: /AllNTUserProfiles)
  If a PatchTextFile section is called with this modifier and the path of the file to be patched contains the constant %UserProfileDir%, the patch section will be executed for all the profiles.
  For a PatchTextFile section which is called from a [ProfileActions] section in the Machine mode the modifier /AllUserProfiles is implied. In the Loginscript Mode the %UserProfileDir% is interpreted as %CurrentProfileDir%.
  (since version 4.11.3.5)

* /encoding <encoding> /since 4.12.4.17 [W/L/M]
  You can add an encoding parameter for the PatchTextFile section. This is needed if the file that you want to modify is not in system encoding. Example:

```
PatchTextFile_my_text "C:/my_file.text" /encoding "utf16le"
```

For allowed encodings see opsi-script encoding

10.6.2. Commands

We have got two commands especially for patching Mozilla preferences files plus the two deprecated and more restricted older versions of these commands:

* Set_Mozilla_Pref ("<preference type>", "<preference key>", "<preference value>")
  sets for <preference type> the value associated with "<preference variable>" to "<preference value>".
  preference type takes any value.
  In current Mozilla preference files there are expressions like
  user_pref("<key>", "<value>")
  pref("<key>", "<value>")
  lock_pref("<key>", "<value>")
  defaultPref("<key>", "<value>")
  lock_pref("<key>", "<value>")
  clearPref("<key>", "<value>")
  Each of them, in fact, any (javascript) function call of the form

functionname (String1, String2)
can be patched with this command by setting the appropriate string for <preference type> (that is, resp. for functionname). If an entry
starting with "functionname (String1" exists in the treated file, it will be patched (and left at its place). Otherwise a new line will be
appended. Unusually in opsi-script, all strings are case sensitive.

• Set_Netscape_User_Pref (<preference variable>, <value>)
sets the line of the given user preference file for the variable <preference variable> to value <value>. The ASCII ordering of the file will
be rebuilt.

(Deprecated!)

• AddStringListElement_To_Mozilla_Pref (<preference type>, <preference variable>, <add value>)
appends an element to a list entry in the given preference file. It is checked if the value that should be added is already contained in the
list (then it will not be added).

• AddStringListElement_To_Netscape_User_Pref (<preference variable>, <add values list>)
(Deprecated!)

The other commands of PatchTextFile sections are not file type specific. All operations are based on the concept that a line pointer exists
which can be moved from top of the file i.e. above the top line down to the bottom (line).

There are three search commands:

• FindLine <search string>
  Finds a line that matches complete (is identic) to <search string>.

• FindLine_StartingWith <search string>
  Finds a line that starts with <search string>.

• FindLine_Containing <search string>
  Finds a line that contains <search string>.

Each command starts searching at the current position of the line pointer. If they find a matching line the line pointer is moved to it.
Otherwise the line pointer keeps its position.
The search is not case sensitive.

<search string> - as all other String references in the following commands - are String surrounded by single or double citation marks.

• GoToTop
  move the line pointer to the top line.

(when we count lines it has to be noted that this commands move the line pointer above the top line). We step any - positive or negative -
number of lines through the file by

• AdvanceLine [line count]
  move the line pointer at [line count] lines forward or backward.

• GoToBottom
  Advancing to the bottom line

By the following command :

• DeleteTheLine
  we delete the line at which the line pointer is directed if there is such a line (if the line pointer has position top, nothing is deleted)

• DeleteAllLines_StartingWith <search string>
  deleting all lines which begin with <search string>

• AddLine <line> or Add_Line <line>
  The line is appended to the file.

• InsertLine <line> or Insert_Line <line>
  <line> is inserted at the position of the line pointer.

• AppendLine <line> or Append_Line <line>
  <line> is appended after the line at which the pointer is directed.
* Append_File <file name>
  reads the file and appends its lines to the edited file.

* Subtract_File <file name>
  removes the beginning lines of the edited file as long as they are identical with the lines of file <file name>.

* SaveToFile <file name>
  writes the edited lines as a file <file name>.

* Sorted
  causes that the edited lines are (ASCII) ordered.

* setValueByKey <keystr> <valuestr> //since 4.11.4.4
  looks for a key/value pair with the key <keystr> and set here as value <valuestr>. If <keystr> not found, the entry will be created at the cursor position.

10.6.3. Examples

For more examples, please check the opsi standard product opsi-script-test and in this product the part $Flag_winst_patch_test_file$ = "on"

10.7. LinkFolder Sections [W/L/M]

In a LinkFolder section start menus entries as well as desktop links are managed.

10.7.1. LinkFolder Sections in Windows

E.g. the following section creates a folder named "acrobat" in the common start menu (shared by all users):

```opsi
[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 a LinkFolder section first must be defined, in which virtual system folder the subsequent instructions are to operate. This expression defines the base folder:

```
set_basefolder <virtual system folder>
```

Virtual system folders to be used are:

```
desktop, sendto, startmenu, startup, programs, desktopdirectory, common_startmenu, common_programs, common_startup, common_desktopdirectory
```

These folders are virtual, for it depends on the operating system (and version), what the resulting physical directory name is.

In the context of standard maschine installations, only the virtual system folders starting with common_ are relevant.

The system folders desktop, sendto, startmenu, startup, programs, desktopdirectory can only be used in the context of a logged on user respectively in a userLoginScript in the context of the opsi extension user Profile Management.

The folders are virtual since the operating system (resp. registry entries) determine the real places of them in the file system. Second, we have to open a subfolder of the selected virtual folder:

```
set_subfolder <folder path>
```
The subfolder name is to be interpreted as a path name with the selected virtual system folder as root. If some link shall be directly placed into the system folder we have to write

```plaintext
set_subfolder ""
```

In the third step, we can start setting links. The command is a multi line expression starting with

```plaintext
set_link
```

and finished by `end_link`.

Between these lines the link parameters are defined in the following format:

```plaintext
set_link
name: [link name]
target: [<complete program path>]
parameters: [command line parameters of the program]
working_dir: [working directory]
icon_file: [icon file path]
icon_index: [position of the icon in the icon file]
shortcut: [keyboard shortcut for calling the target]
end_link
```

The `target` name is the only essential entry. The other entries have default values:

- `name` defaults to the program name.
- `parameters` defaults to an empty string.
- `icon_file` defaults to the `target`.
- `icon_index` defaults to 0.
- `shortcut` defaults to empty. // since 4.11.6.7

`shortcut` may be a combination of [shift, alt, ctrl] (not case sensitive) divided by " " (Space), "," (minus char), "+" (plus char) and 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 their Virtual Key Code identifier. To get these identifiers (as well as the allowed combinations) just use the following helper program:

```plaintext
http://download.uib.de/opsi4.0/helper/showkeys.exe
```

Keep in mind that a `shortcut` references the keys and not their country specific layout. The Key `VK_OEM_3` is on an English keyboard the char ; and on a German the letter Ö.

Examples for allowed shortcuts:

- O (The Key O)
- VK_O (The Key O)
- Ctrl-O (The combination Ctrl O)
- Ctrl-Alt-Shift-O (The combination Ctrl Alt Shift O)
- Ctrl+Alt+Shift+0 (The combination Ctrl Alt Shift O)
- Ctrl Alt Shift O (The combination Ctrl Alt Shift O)
- Ctrl-Alt-Shift-VK_O (The combination Ctrl Alt Shift O)
- Ctrl-Alt-Shift-VK_F12 (The combination Ctrl Alt Shift F12)

If the referenced target does not lie on an mounted share at the moment of link creation Windows shortens its name to the 8.3 format.

**Workaround:**
Create a correct link when the share is connected.
Copy the ready link file to a location which exists at script runtime.
Let this file be the target.

- `delete_element <Linkname>`
  remove a link from the open folder.

- `delete_subfolder <Folderpath>`
  folder is removed from the base virtual folder
10.7.2. Examples

```plaintext
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
```

produces the following log:

```plaintext
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
```

For more examples, please check the opsi standard product `opsi-script-test` and in this product the part `$Flag_winst_link_folder$ = "on"`.

10.7.3. LinkFolder-Sections in Linux

LinkFolder sections are supported also on Linux since version 4.11.5.2.

Possible bas folders are:
- `common_programs`, `common_autostart`, `desktop`, `autostart`
- Subfolder is always "" (empty).

The Link Option `icon_index` is ignored.

As additional Link Option we have: `link_categories`.

Here you may use the following values seperated and terminated by a semicolon:

The LinkFolder Sektion will work at Linux with different Desktop systems.
10.8. XML2 Section [W/L/M]

A popular way to keep configuration data or data at all is a file in XML document format. Its syntax follows the conventions as defined in the XML (or "Extended Markup Language") specification (http://www.w3.org/TR/xml/).

**opsi-script** offers two different ways to handle XML files:

- The xml2 sections that are described in this chapter  
  Since opsi-script version 4.12.1  
- The deprecated, still working (but only at windows) XMLPatch sections (XMLPatch Sections) and functions which are more powerful but also more complicated than the newer xml2 methods. So we recommend to use the xml2 sections and functions.

The xml2 implementation is divided in

- the xml2 section as described over here, with the goal to make it easy to manipulate xml data  
- the xml2 functions with the target to analyze given xml data  
  see also : XML related functions (XML2)  
  see also : XML2 Functions

### 10.8.1. XML structure and wording

Let's have a look at a simple xml file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rootnode>
  <node_level-1_number-1>
    <node_level-2_A color="blue">Hello World</node_level-2_A>
    <node_level-2_B color="green" count="65">
    </node_level-2_B>
    <node_level-2_C>
    </node_level-2_C>
  </node_level-1_number-1>
  <node_level-1_number-2>
  </node_level-1_number-2>
</rootnode>
```

To describe the structure in this xml file we use the following wording:

- **xml file**  
  A file that contains xml data.

- **xml header**  
  XML meta data at the beginning of a xml file. In our example:  
  ```xml
  <?xml version="1.0" encoding="UTF-8"?>
  ```

- **node**  
  The xml node starts with a open element `<` followed by a identifier and `>` and ends with the close element `<\` followed by the same identifier and `>` Example:  
  ```xml
  <mynode>
  </mynode>
  ```
  
  If (like in the example above) there is no additional information this node can be written as:  
  ```xml
  <mynode>
  </mynode>
  ```
  
  In the open element the identifier may be followed by one or more attributes.  
  Between the open and the close element you may find the nodetext.

- **root node**  
  The base node of the xml tree. In our example:  
  ```xml
  <rootnode>
  </rootnode>
  ```

- **attribute** is a key/value pair that is part of the open element and comes after the node identifier. like:  
  ```xml
  color="blue" in <node_level-2_A color="blue">
  ```

- **nodetext**  
  is text that may come between the open and the close element. Like:  
  ```xml
  Hello World in <node_level-2_A color="blue">Hello World</node_level-2_A>
  ```
xml2path
is a opsi xml2 specific notation to give a path throug the xml tree.
Example: `<node_level-1_number-1> // </node_level-2_B>`
It is the sequence of nodes below the root node

xml2stringlist
The opsi-script xml2 functions do not work directly on a xml file, but on a stringlist representation of this file or parts of it (a node).
So with the function getXml2DocumentFromFile(<path to xml file>) you get a stringlist that contains the representation of the content of this file in a xml2stringlist. This variable may be used to analyze the xml2stringlist by other xml2 functions.
The result of those functions may be also a stringlist from type xml2stringlist
In fact the the xml2stringlist is a string list that contains the content of the xml file in a special format and without the header. But do not try to construct it without using getXml2DocumentFromFile or getXml2Document.
see : XML2 Functions

10.8.2. CallParameter

The name of the file to be patched is passed as a parameter.

Example:
xml2_test "%scriptpath%\dummy.xml"

If the given file does not exist, it will be created. While creating the file as name of the root node we use the value of the command
rootNodeOnCreate = <node name>. If this command is missing in the section the fall back is the root node name: rootNode. (since 4.12.4.27)

There are optional modifiers:

1. /AllUserProfiles // since 4.12.4.27
   If a XML2 section is called with this modifier, then all directories under %UserProfileDir% will be patched, which means that this patch is performed for all user profiles.
   When a XML2 section is called within a [ProfileActions] section, then the modifier /AllUserProfiles is implicit. In logscript mode, %UserProfileDir% will be interpreted as %CurrentProfileDir%.

2. /encoding <encoding> //since 4.12.4.27 [W/L/M]
   By default the given XML file is expected in the encoding "UTF-8".
   You can add an encoding parameter for the XML2 section. This is needed if the file that you want to modify is not in "UTF-8" encoding.
   Example :
   `XML2_my_xml "C:/my_file.xml" /encoding "utf16le"
``

For allowed encodings see opsi-script encoding

10.8.3. Commands

The idea of the syntax here is oriented at the command syntax of other patch sections like registry or patches in `opsi-script.

There exist the following commands:

1. strictMode = (true/false) : Default: false
2. openNode <xml2 path>
3. SetAttribute <attr name> <attr value>
4. AddAttribute <attr name> <attr value>
5. DeleteAttribute <attr name>
6. addNewNode <node name>
7. setNodeText <string>
8. DeleteNode <xml2 path>
In detail:

The first step is to navigate to the node where we like to change things.

`strictMode = (true/false); Default: false`

`openNode <xml2 path>`
Open the given Path as actual node. If the path does not exist it will be created

`DeleteNode <xml2 path>`

The `<xml2 path>` is the path to our target node. It may have two different forms, according to the value of `strictMode`:

- `<xml2 path> strictMode =false` (Default):
  A line of xml node names only with no attributes separated by `//`
  Example:
  `node_level-1_number-1 // node_level-2_B`

- `<xml2 path> strictMode =true`:
  A line of xml node names with all existing attributes separated by `//`
  Example: `node_level-1_number-1 // node_level-2_B color="green" count="65"`

All other commands operate on an opened xml node.

- `SetAttribute <attr name> <attr value>`
  At the actual node, set `<attr value>` as value of `<attr name>`. If `<attr name>` not exists, it will be created.

- `AddAttribute <attr name> <attr value>`
  If at the actual node the attribute `<attr name>` not exists, it will be created with `<attr value>` as value. If `<attr name>` still exists, nothing will be changed.

- `DeleteAttribute <attr name>`
  If at the actual node the attribute `<attr name>` exists, it will be deleted.

- `addNewNode <node name>`
  Create at the actual node a new sub node `<node name>` and make this new node to the actual node.

- `setNodeText <string>`
  Set `<string>` as the `nodetext` of the actual node.

- `gotoParentNode`
  Make the parent node to the actual node.

- `* setNodePair <keyNodeName> <keyNodeTextContent> <valueNodeName> <valueNodeTextContent>` // since 4.12.4.28
  Creates a `<dict>` entry like it is used in the Apple info.plist files:

  ```xml
  <dict>
  <key>CFBundleExecutable</key>
  <string>opsi-script</string>
  <key>CFBundleIdentifier</key>
  <string>org.opsi.opsi-script</string>
  <key>CFBundleName</key>
  <string>opsi-script</string>
  <key>CFBundleShortVersionString</key>
  <string>4.12.4.35</string>
  </dict>
  ```

Example:

```xml
setNodePair "key" "CFBundleShortVersionString" "string" "4.12.4.35"
```
10.8.4. XML2 Examples

We assume we have file `dummy.xml` with the content:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rootnode>
    <node_level-1_number-1>
        <node_level-2_A color="blue">Hello World</node_level-2_A>
        <node_level-2_B color="green" count="65">
            <node_level-2_C>
                </node_level-2_C>
        </node_level-2_B>
    </node_level-1_number-2>
</node_level-1_number-1>
</rootnode>
```

The following code:

```bash
comment "Testing: 
message "opennode not existing node"
set $xml2strictMode$ = 'false'
set $xml2nodepath$ = 'node_level-1_number-1 // node_level-2_B // node_level-3_A'
set $xml2changeValue$ = '"color" "yellow"'
set $xml2cmdLine1$ = "strictMode = "$xml2strictMode"
set $xml2cmdLine2$ = "openNode "$xml2nodepath$"
set $xml2cmdLine3$ = "setAttribute "$xml2changeValue"
XML2_dummy_xml $HomeTestFiles$+"dummy.xml"
set $ConstTest$ = "yellow"
set $list1$ = loadTextFile($HomeTestFiles$+"dummy.xml")
set $tmp$ = takeFirstStringContaining($list1$,"node_level-3_A")
set $CompValue$ = takeString(1, splitString ($tmp$, '"'))
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
set $xml2cmdLine4$ = "getXml2DocumentFromFile"
set $list1$ = getXml2DocumentFromFile($HomeTestFiles$+"dummy.xml")
set $list2$ = xml2GetFirstChildNodeByName($list1$,"node_level-3_A")
set $CompValue$ = xml2GetAttributeValueByKey($list2$,"color")
if ($ConstTest$ = $CompValue$)
    comment "passed"
else
    set $TestResult$ = "not o.k."
    LogWarning "failed"
endif
```

[XML2_dummy.xml]
(xml2cmdLine1)
(xml2cmdLine2)
(xml2cmdLine3)
(xml2cmdLine4)
(xml2cmdLine5)
(xml2cmdLine6)
(xml2cmdLine7)
(xml2cmdLine8)
(xml2cmdLine9)
message "opennode not existing node"
;
XML2_dummy_xml $HomeTestFiles$ + "\dummy.xml"
;
; Test 1
set $ConstTest$ = "yellow"
set $list1$ = loadTextFile($HomeTestFiles$ + "\dummy.xml")
set $tmp$ = takeFirstStringContaining($list1$, "node_level-3_A")
set $CompValue$ = takeString(1, splitString($tmp$, ""))
if ($ConstTest$ = $CompValue$)
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif
;
; Test 2
set $ConstTest$ = "yellow"
set $list1$ = getXml2DocumentFromFile($HomeTestFiles$ + "\dummy.xml")
set $list2$ = xml2GetFirstChildNodeByName($list1$, "node_level-3_A")
set $CompValue$ = getXml2AttributeValueByKey($list2$, "color")
if ($ConstTest$ = $CompValue$)
  comment "passed"
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif

[XML2_dummy_xml]
strictMode = false
openNode 'node_level-1_number-1 // node_level-2_B // node_level-3_A'
SetAttribute "color" "yellow"

produces e.g the log:

message opennode not existing node
Set $xml2nodepath$ = "node_level-1_number-1 // node_level-2_B // node_level-3_A"
The value of the variable "$xml2nodepath$" is now: "node_level-1_number-1 // node_level-2_B // node_level-3_A"
Set $xml2changeValue$ = ""color" "yellow"
The value of the variable "$xml2changeValue$" is now: ""color" "yellow"
Set $xml2cmdLine1$ = "strictMode = $xml2strictMode$"
The value of the variable "$xml2cmdLine1$" is now: "strictMode = false"
Set $xml2cmdLine2$ = "openNode $xml2nodepath$"
The value of the variable "$xml2cmdLine2$" is now: "openNode "node_level-1_number-1 // node_level-2_B // node_level-3_A"
Set $xml2cmdLine3$ = "SetAttribute "$xml2changeValue$"
The value of the variable "$xml2cmdLine3$" is now: "SetAttribute "color" "yellow"

try to open File: c:\opsi.org\tmp\testFiles\dummy.xml
try to load File: c:\opsi.org\tmp\testFiles\dummy.xml
File: c:\opsi.org\tmp\testFiles\dummy.xml read
success: create xmldoc from file: c:\opsi.org\tmp\testFiles\dummy.xml
StrictMode is set to : False
We will OpenNode : node_level-1_number-1 // node_level-2_B // node_level-3_A
begin to open nodepath : node_level-1_number-1 // node_level-2_B // node_level-3_A
-- pathes.Count: 3
path element 1 : node_level-1_number-1
thisnodename
leavingPath node_level-1_number-1
node 1: nodename node_level-1_number-1
begin to get node nodename: node_level-1_number-1 with attributes:
  Found node 1: nodename: node_level-1_number-1
path element 2 : node_level-2_B
thisnodename
leavingPath node_level-2_B
node 2: nodename node_level-2_B
begin to get node nodename: node_level-2_B with attributes:
  Found node 2: nodename: node_level-2_B
path element 3 : node_level-3_A
thisnodename
leavingPath node_level-3_A
node 3: nodename node_level-3_A
begin to get node nodename: node_level-3_A with attributes:
opennode: node not found 3: nodename: node_level-3_A
actNode=nil: opennode: node not found, maybe 3: nodename: node_level-3_A
nodepath does not exists - try to create: node_level-1_number-1 // node_level-2_B // node_level-3_A
begin to make node with path: node_level-1_number-1 // node_level-2_B // node_level-3_A and TEXT_CONTENT:
actNodeSet <> nil
begin to open nodepath : node_level-1_number-1 // node_level-2_B // node_level-3_A
-- pathes.Count: 3
path element 1 : node_level-1_number-1
actnode: rootnode
begin to get node nodename: node_level-1_number-1 with attributes:
ode(s) found with name node_level-1_number-1: 1
1 -> find attributes for node `node_level-1_number-1`, number of attributes: 0

all attributes have to fit, nodename: `node_level-1_number-1`

actNodeSet after retrieving key/value

```
actNodeSet:
  node 0: elementname: "node_level-1_number-1"

Non-null element(s) in act node set: 1
result: true, actNode and newnode is `node_level-1_number-1`

Found node 1: nodename: `node_level-1_number-1`
path element 2: `node_level-2_B`

thisnodename

leavingPath `node_level-2_B`
node 2: nodename: `node_level-2_B`
actnode: `node_level-1_number-1`

begin to get node nodename: `node_level-2_B` with attributes:
node(s) found with name `node_level-2_B`: 1

1 -> find attributes for node `node_level-2_B`, number of attributes: 0

all attributes have to fit, nodename: `node_level-2_B`

Attribute count mismatch: given by path: 0 but node has: 2

actNodeSet after retrieving key/value

```
Non-null element(s) in act node set: 0
result: false, actnode is nil, length of actNodeSet is 0

makeNodePathWithTextContent: node not found 2: nodename: `node_level-2_B`,
Node will be created
begin to make node with nodename: `node_level-2_B`

path element 3: `node_level-3_A`

thisnodename

leavingPath `node_level-3_A`
node 3: nodename: `node_level-3_A`
actnode: `node_level-2_B`

makeNodePathWithTextContent: node not found 3: nodename: `node_level-3_A`,
Node will be created
begin to make node with nodename: `node_level-3_A`

actNode know node 3: nodename: `node_level-3_A`

successfully created nodepath: `node_level-1_number-1 // node_level-2_B // node_level-3_A`

We will setAttribute : color : yellow
begin setAttribute name: color, value: yellow
setAttribute, create attribute with name: color value: yellow

```

```

successfully setAttribute : color : yellow
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

try to open file: c:\opsi.org\tmp\testFiles\dummy.xml
file saved: c:\opsi.org\tmp\testFiles\dummy.xml
successful written xml doc to file: c:\opsi.org\tmp\testFiles\dummy.xml

Set $ConstTest$ = "yellow"
The value of the variable "$ConstTest" is now: "yellow"
Set $list1$ = loadTextFile($HomeTestFiles$+"\dummy.xml")
The value of the variable "$list1" is now:

```

```

```

```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
The following code:

```plaintext
message "addNewNode"
set $xml2strictMode$ = 'false'
set $xml2nodepath$ = "node_level-1_number-1 // node_level-2_C"
set $xml2changeValue$ = "node_level-3_C"
set $xml2cmdLine10$ = "openNode "$xml2nodepath$"
set $xml2cmdLine40$ = "SetAttribute "node" "new"
XML2_dummy_xml $HomeTestFiles$+"dummy.xml"
set $ConstTest$ = '<node_level-3_C node="new"/>
set $list1$ = loadTextFile($HomeTestFiles$+"dummy.xml")
set $tmp$ = takeFirstStringContaining($list1$,"node_level-3_C")
set $CompValue$ = Trim($tmp$)
if ($ConstTest$ = $CompValue$)
  comment: passed
else
  set $TestResult$ = "not o.k."
  LogWarning "addNewNode failed"
endif
set $ConstTest$ = "new"
set $list1$ = getXml2DocumentFromFile($HomeTestFiles$+"dummy.xml")
set $list2$ = xml2GetFirstChildNodeByName($list1$,"node_level-3_C")
set $CompValue$ = getXml2AttributeValueByKey($list2$,"node")
if ($ConstTest$ = $CompValue$)
  comment: passed
else
  set $TestResult$ = "not o.k."
  LogWarning "failed"
endif
```

produces e.g the log:

```plaintext
message addNewNode
Set $xml2nodepath$ = "node_level-1_number-1 // node_level-2_C"
The value of the variable "$xml2nodepath$" is now: "node_level-1_number-1 // node_level-2_C"
Set $xml2changeValue$ = "node_level-3_C"
The value of the variable "$xml2changeValue$" is now: "node_level-3_C"
Set $xml2cmdLine10$ = "openNode "$xml2nodepath$"
The value of the variable "$xml2cmdLine10$" is now: "openNode "node_level-1_number-1 // node_level-2_C"
Set $xml2cmdLine40$ = "SetAttribute "node" "new"
The value of the variable "$xml2cmdLine40$" is now: "SetAttribute "node" "new"
       try to open File: c:\opsi.org\tmp\testFiles\dummy.xml
       try to load File: c:\opsi.org\tmp\testFiles\dummy.xml
File: c:\opsi.org\tmp\testFiles\dummy.xml read
  success: create xmldoc from file: c:\opsi.org\tmp\testFiles\dummy.xml
  StrictMode is set to : True
  We will OpenNode : node_level-1_number-1 // node_level-2_C
  begin to open nodepath : node_level-1_number-1 // node_level-2_C
     -- pathes.Count: 2
       path element 1 : node_level-1_number-1
          thisnodename
          leavingPath node_level-1_number-1
       node 1: nodename node_level-1_number-1
          begin to get node node: node_level-1_number-1 with attributes:
          node(s) found with name node_level-1_number-1: 1
            1 -> find attributes for node node_level-1_number-1, number of attributes 0
               all attributes have to fit, nodename node_level-1_number-1
               ancnodeset after retrieving key/value
```
For further examples see the product opsi-script-test especially the file sub-scripts/xml2test.opsiscript
10.9. XMLPatch Sections [W]

Warning: This section is deprecated. It will be not removed, but it is frozen and not under further development. Also this section is Windows only and will never be ported to any other OS.
We recommend to use the XML2 Section and XML2 Functions instead.

Today, the most popular way to keep configuration data or data at all is a file in XML document format. Its syntax follows the conventions as defined in the XML (or "Extended Markup Language") specification (http://www.w3.org/TR/xml/).

opsi-script offers XMLPatch sections for editing XML documents.

With the actions defined for this section type opsi-script can

- select (and optionally create) sets of elements of a XML document according to a path description
- patch all elements of a selected element set
- return the names and/or attributes of the selected elements to the calling section

10.9.1. Parameter

When calling an XMLPatch section the document path name is given as parameter, e.g.

XMLPatch_mozilla_mimetypes $mozillaprofilepath$ + "\mimetypes.rdf"

10.9.2. Structure of a XML Document

A XML document logically describes a "tree" which starting from a "root" - therefore named document root- grows into branches. Every branch is labelled a node. The sub nodes of some node are called children or child nodes of their parent node.

In XML, the tree is constructed from elements. The beginning of any element description is marked by a tag (similarly as in HTML) i.e. a specific piece of text which is set into a pair of angle brackets ("<" "">"), The end of the element description is defined by the the same tag text but now bracket by "<" and "">". If an element has no subordinated elements then there is no space needed between start tag and end tag. In this case the two tags can be combined to one with end bracket "/>".

This sketch shows a simple "V"-tree - just one branching at the root level, rotated so that the root is top: .- | root node (level 0) /

node 1 and node 2 both on level 1 . . implicitly given end nodes below level 1 .- This tree could be described in XML in the following way:

```xml
<?xml version="1.0"?>
<root>
  <node_level_1_no_1>
  </node_level_1_no_1>
  <node_level_1_no_2>
  </node_level_1_no_2>
</root>
```

The first line has to declare the XML version used. The rest of lines describe the tree.

So long the structure seems to be simple. But yet we have only "main nodes" each defining an element of the tree and marked by a pair of tags. But each main node may have subnodes of several kinds.

Of course, an element may have subordered elements, e.g. we may have subnodes A to C of node 1:

```xml
<nodes_level_1_no_1>
  <node_level_1_2_A>
  </node_level_1_2_A>
  <node_level_1_2_B>
  </node_level_1_2_B>
  <node_level_1_2_C>
  </node_level_1_2_C>
</nodes_level_1_no_1>
```

If there are no subordinated elements an element can have subordinated text. Then it is said that the element has a subordinated text node.

Example
A line break placed in the text node is now interpreted as part of the text where otherwise it is only a means of displaying XML structure. To avoid a line break belonging to "hello world" we have to write

```
<node_level_1_no_2>hello world</node_level_1_no_2>
```

Every element (no matter if it has subordinated elements or subordinated text) is constituted as a main node with specific tags. It can be further specified by attributes, so called attribute nodes. For example, there may be attributes "colour" or "angle" that distinguish different nodes of level 1.

```
<node_level_1_no_1 colour="green" angle="65" />
```

For selecting a set of elements any kind of information can be used:

1. the element level,
2. the element names that are traversed when descending the tree (the "XML path"),
3. names and values of the used attributes,
4. the ordering of attributes,
5. the ordering of elements,
6. other relationships of elements,
7. the textual content of elements (resp. their subordinated text nodes).

In opsí-script, selection based on criteria (1) to (3) and (7) is implemented.

### 10.9.3. Options for Selection a Set of Elements

Before any operation on the contents of a XML file the precise set of elements has to be determined on which it will be operated. The set is constructed step by step by defining the allowed paths through the XML tree. The finally remaining end points of the paths define the selected set.

The basic opsí-script command is

- **OpenNodeSet**

There are two formats for defining the allowed paths a short and a long format.

**Explicit Syntax**

The more explicit syntax may be seen in the following example (for a more complex example Section 12.4, “XML File Patching: Setting Template Path for OpenOffice.org 2”):

```
openNodeSet
document.root
 all_childelements_with:
  elementname:"define"
 all_childelements_with:
   elementname:"handler"
   attribute: extension value="doc"
 all_childelements_with:
   elementname:"application"
end
```

**Short Syntax**

The same node set is given by the line:
In this syntax, the slash separates the steps into the tree structure which are denoted in the more explicit syntax each by an own description.

**Selecting by Textual Content (only for explicit syntax)**

Given the explicit syntax we may select elements by the textual content of elements:

```xml
openNodeSet
document root
all_childelements_with:
  all_childelements_with:
    elementname: "description"
    attribute: "name" value="mozilla"
  all_childelements_with:
    elementname: "linkurl"
text: "http://www.mozilla.org"
end
```

**Parametrizing Search Strategy**

In the exemplary descriptions of XML tree traversals there remain several questions.

- Shall an element be accepted if the element name and the listed attributes match but other attributes exist?
- Is the search meant to give one single result value, that is should the resulting element set have no more than one element (and otherwise, the XML file is to considered as erroneous)?
- Conversely, is it meant that a traversal shall at any rate lead to some result, i.e. do we have to create the element if no matching element exists?

To answer these questions explicitly there are parameters for the OpenNodeSet command. The following lines show the default settings which can be varied by changing the Boolean values:

```xml
- error_when_no_node_existing false
- warning_when_no_node_existing true
- error_when_nodecount_greater_1 false
- warning_when_nodecount_greater_1 false
- create_when_node_not_existing false
- attributes_strict false
```

With short syntax, parametrizing precedes the OpenNodeSet command and holds for all levels of the XML tree. With the explicit syntax the parameters may be set directly after the OpenNodeSet command or be newly set for each level. In particular the option „create when node not existing“ may be set for some levels but not for all.

**10.9.4. Patch Actions**

There exists a bundle of commands which operate on a selected element set

- for setting and removing attributes
- for removing elements
- for text setting.

In detail:

- **SetAttribute** "attribute name" value="attribute value"
  sets the specified attribute for each element in the opened set to the specified value. In the attribute does not exist it will be created. Example: **SetAttribute** "name" value="OpenOffice Writer"

On the contrary, the command

- **AddAttribute** "attribute name" value="attribute value"
sets the specified attribute only to the specified value if it does not exists beforehand. An existing attribute keeps its value. E.g. the command

```
AddAttribute "name" value="OpenOffice Writer"
```

would not overwrite the value if there was named another program before.

By

```
* DeleteAttribute "attribute name"
```

we remove the specified attribute from each element of the selected element set.

The command

```
* DeleteElement "element name"
```

removes all elements with main node name (tag name) element name from the opened element set.

Finally there exist two commands for setting resp. adding text nodes:

- **SetText “Text”**

and

- **AddText “Text”**

Example:

```
SetText "rtf"
```

transforms the element

```
<fileExtensions>doc</fileExtensions>
```

into

```
<fileExtensions>rtf</fileExtensions>
```

By

```
SetText ""
```

we remove the text node completely.

The variant

```
AddText "rtf"
```

sets the text only if there was no text node given.

### 10.9.5. Returning Lists to the Caller

A XMLPatch section may return the retrieved informations to the calling primary section. The result always is a String list, and to get it, the call must done via the String list function `getReturnListFromSection`. E.g. we may have the following String list setting in an Actions section where we use a XMLPatch_mime section:

```plaintext
DefStringList $list1$
set $list1$=getReturnListFromSection ('XMLPatch_mime "c:\mimetypes.rdf"')
```

Inside the XMLPatch section we have `return` commands that determine the content of returned String list:

- **return elements** fills the selected elements completely (element name and attributes) into the return list.
- **return attributes** produces a list of the attributes.
- **return elementnames** produces a list of the element names.
- **return attributenames** gives a list only of the attribute names.
- **return text** list all textual content of the selected elements.
- **return counting** gives a report with numerical informations: line 0 contains the number of selected elements, line 1 the number of attributes.
10.9.6. Examples

For further examples see the product opsi-script-test especially the sector with $Flag_winst_xml$ = "on"

10.10. ProgmanGroups Sections

This section type is deprecated.

10.11. WinBatch-Sections [W/L/M]

In a WinBatch section any windows executable can be started.
E.g. we may start some existing setup program by the following line in a WinBatch section

```
[winbatch_install]
"%scriptpath%\setup.exe"
```

Winbatch section are designned to start programs (*.exe) directly.
To call data files that are connected to programs is deprecated but still supported. If you do this you will get a deprecated warning. Example:
ok: notepad.exe test.txt
deprecated (not ok): test.txt

10.11.1. Call Parameter (Modifier)

There are several parameters of the WinBatch call which determine if (or how long) opsi-script shall be wait for the started programs returning.

- **/WaitOnClose**
  Is the default
  opsi-script waits for every initiated process to come back.

- **/LetThemGo**
  This is the contrary to /WaitOnClose. It is used if opsi-script shall proceed while the started processes run in their own threads.

- **/WaitSeconds [number of seconds]**
  If a call includes the parameter /WaitSeconds [number of seconds], then opsi-script is waiting for [number of seconds] before proceeding. In the default configuration, we also wait for any programs that are currently running to finish. If we combine the parameter /WaitSeconds with the option /LetThemGo, then opsi-script continues processing after the waiting time is finished.

- **/WaitForProcessEnding <program name>**
  Waits for the process called <program name> to end.
  Should be combined with /TimeOutSeconds.

Explanation:
When starting an external process from a winbatch call, the opsi-script waits for the current process to finish before executing the next command in the script.

- **/32Bit //since 4.11.3.5 [W]**
  This is the default. The paths within the section are assumed to be 32 bit paths.
  Example: c:\windows\system32\regedit.exe calls (even when running on a 64 bit system) the 32 bit regedit.exe.

- **/64Bit //since 4.11.3.5 [W]**
  The paths within the section are assumed to be 64 bit paths.
  Example: c:\windows\system32\regedit.exe executes (running on a 64 bit system) the 64 bit regedit.exe.

- **/SysNative //since 4.11.3.5 [W]**
  The paths within the section are assigned according to the OS architecture interpreted.
  Example: c:\windows\system32\regedit.exe running on a 64bit system calls the 64 bit regedit.exe and running on a 32 bit system the 32 bit regedit.exe.

Example:
** /RunAsLoggedonUser //since 4.11.3.5 [W]

This is available only in the context of userLoginScripts. The program is executed as the user, who has just logged on. This modifier has the following limitation:
- insufficient tested on NT6 and possibly of limited effect.

** Figure 6. Sequential processing of a script that waits for the end of a program**

There are some external programs which start another process and then end without waiting for their child process to end. From the point of view of opsi-script, the process is ended and the next command could be started.
If you run an uninstall program and a setup program in sequence and the uninstall program works with such a child process, you can have conflicting processes running because the uninstallation and installation processes are running at the same time.

Using the modifier `/WaitForProcessEnding` helps to avoid such a situation.

- `/TimeOutSeconds <seconds>`
  A timeout setting. After waiting `<seconds>`, `opsi-script` will end the process.
  Since version 4.11.3, `/TimeOutSeconds` may be used without a waiting condition (e.g. `/WaitForProcessEnding`) but not in combination with `/WaitSeconds`.
  Since version 4.11.4.6 the time progress from start until timeout is displayed by the progressbar.

Example:
Winbatch\uninstall /WaitForProcessEnding "uninstall.exe" /TimeOutSeconds 20

[Winbatch\uninstall]
"%ScriptPath%\uninstall_starter.exe"

• /RunElevated [W]
  Starts a process that has a security token with elevated privileges. This modifier has the following restrictions:
  - For NT5 it does not change anything.
  - A process started with this modifier has no network access. So you should copy a program to a temporary local directory, but do not start it from a network share.
  - You may see problems while using the graphical interface. Therefore true silent installations are the better choice in this case.
  - Functions only in the context of ops\script.

• getLastExitCode
  Returns a string that contains the value of the exitcode of the the process that was last called by a WinBatch / DosBatch / ExecWith section.
  When using a DosBatch or ExecWith section, you will normally get the exitcode of the interpreter that was called. To get the exitcode of your script you have to define it explicitly.

• /RunAsLoggedOnUser // since 4.11.3.5 [W] ; works only inside userLoginScripts

• /32Bit or /64Bit or /SysNative //since 4.11.3.5 [W]
  These modifiers control if the path to a called program is interpreted as 32 or 64 Bit Path. So if want for example call a %system%\cmd.exe you call a 32 bit program by default. If you use the /64bit modifier you will get with the same call the 64 bit version.

• /WaitForWindowAppearing [window title] [W]
 resp.
  /WaitForWindowVanish [window title] [W]
  Both are deprecated. Please use /WaitForProcessEnding

The first option means that ops\script waits until any process lets pop up a window with title window title. With the second option ops\script is waiting as long as a certain window (1) appeared on the desktop and (2) disappeared again.

CAUTION: These options only know windows of 32-bit programs

10.11.2. Examples

For further examples see the product ops\script-test especially the sector with $Flag_winst_winbatch$ = "on"

10.12. DOSBatch/DosInAnIcon (ShellBatch/ShellInAnIcon) Sections [W/L/M]

Via DOSBatch (also called ShellBatch) sections a ops\script script uses Windows shell scripts for tasks which cannot be fulfilled by internal commands or for which already a batch script solution exists.

ops\script waits until the DOS-batch ends, before it is proceeding with the next script-section.

A DOSBatch section is simply processed by writing the lines of the sections into the file _opsiscript<random>.cmd in c:\opsi.org\tmp and then calling this file in the context of a cmd.exe shell. This explains that a DosBatch section may contain all Windows shell commands can be used.

Compared with calling a cmd-file in a WinBatch section, the DOSBatch section presents certain advantages:

• ops\script variables or constants in the section can be easily used because they will be substituted before execution.
• The output of a DosInAnIcon/ShellInAnIcon call at Windows is written to the log file.
• The output of a DosInAnIcon/ShellInAnIcon or DosBatch/ShellBatch call at Linux is written to the log file.
• The output of a DosInAnIcon/ShellInAnIcon call is written to an output window if the section is called with the parameter /showoutput.
• The output of the shell commands can be captured by using the String list function.

The section type *DOSInAnIcon* or *ShellInAnIcon* is identical to *DOSBatch/ShellBatch* regarding syntax and execution method but has a different appearance:

For *DOSInAnIcon/ShellInAnIcon*, a shell process is created with view set to minimized. That has the consequence that it is executed “in an icon”. No command window appears, user interaction is suppressed. The output of the call is written to the log file.

Don’t use commands that wait for user interaction.

### 10.12.1. Parameter

There are two kinds of parameters which may be passed to the section call:

- Parameters which are directly passed to the called batch file.
- Parameter which modify the way *opsi-script* will handle the section.

The calling syntax is:

```plaintext
Sektionsname [batch parameter] [winst [modifier]]
```

Possible opsi-script modifier are (since 4.11.1):

- `/32bit`
- `/64bit`
- `/Sysnative`
- `/showoutput // since 4.11.6`

* `/encoding <encoding> //since 4.12.4.17 [W/L/M]`

By default the content of the section will be stored to a temporary file in system encoding. This is in the most cases the best choice. If you really knows what you are doing, you may change the encoding of the stored temporary file:

You can add an encoding parameter after the WINST keyword.

Example:

```plaintext
DosInAnIcon_do_encoding_example WINST /encoding "utf8"
```

For allowed encodings see *opsi-script encoding*

Since 4.12.4.0 the following modifiers may be used that are well known from *winbatch* sections and are explained over there: **Call Parameter (Modifier)**

- `/runElevated [W]`
- `/TimeoutSeconds <number>`
- `/WaitForProcessEnding <string>`
- `/LetThemGo`

These opsi-script modifier have to be separated by the key word **winst** from the other parameters.

Other parameters of a *DosBatch* section are directly passed as quasi command line parameters to the Windows shell script.

For example, we may call *DosBatch_1* in Actions section to get a “Hello World” from the DOS echo command:
the execution of the Dos-Batch command echo with parameters *today we say "Hello World"*. 

The next example will be on a 64 bit system executed in a 64 bit cmd.exe and gives the output *today we say*:

Since Version 4.11.5 not only string constants but also string variables are allowed as parameters (but no string functions)

Example:

Code from opsi-script-test:

```bash
comment "Testing parameters for ShellBatch"
set $ConstTest$ = "Hello world"
set $list$ = getOutStreamFromSection('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 $list$ = getOutStreamFromSection('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
```

produce the log:
10.12.2. Catch the output

The output of the shell commands can be captured by using the string list function `getOutStreamFromSection()` from the `opsis-script -scripts main-section` see also: `Simple String Values generated from String Lists or Files`.

If the return list shall be evaluated programmatically it is advised to use the `@` prefix of commands. Such we suppress the repetition of the command line in the output which may different formats dependent on system configurations.
10.12.3. Examples

For further examples see the product opsi-script-test and there look at $Flag_winst_dos$ = "on"

10.13. Registry-Sections [W]

By a Registry section call we can create, patch and delete entries in the Windows registry. As usual, opsi-script logs every operation in detail as long as logging is not turned off.

10.13.1. Examples

Let us set some registry variables by a call to the section Registry_TestPatch where the section is given by

```
[Registry_TestPatch]
opentkey [HKEY_Current_User\Environment\Test]
set "Testvar1" = "c:\rutils;%Systemroot%\hey"
set "Testvar2" = REG_DWORD:0001
```

For further examples see the product opsi-script-test and there look at $Flag_subregistry$ = "on"

10.13.2. Call Parameters

- The standard call of a Registry section has no parameters. This is sufficient as long as the operations aim at the standard registry of a Windows system and all entries can be defined using a globally defined registry path.

- `/AllNTUserDats`
  opsi-script also offers that the patch commands of a Registry section are automatically executed "for all users" which are locally defined. I.e. the patches are made for all user branches of the local registry. This interpretation of the section is evoked by the parameter `/AllNTUserDats`

Further parameters control which syntactical variant of the Registry section shall be valid:

- `/regedit`
  The parameter `/regedit` declares that the syntax corresponds the export file syntax of the Windows Registry Editor regedit. Such, the lines of a regedit export file may directly be used as a Registry resp. the file itself can serve as an external section (see Section 10.13.5, "Registry Sections in Regedit Format").

- `/addReg`
  Similarly, the parameter `/addReg` declares that the Registry section syntax is that of an inf-file (as used e.g. for driver installations) (see Section 10.13.6, "Registry Sections in AddReg Format").

These not opsi-script specific syntactical variants are not defined in this manual since they usually will be generated programmatically.

There are also the Options:

- `/32Bit`
- `/64Bit`
- `/SysNative`

which manipulate the registry write redirection on 64 Bit systems. (see Chapter 11, 64 Bit Support on Windows [W]).

10.13.3. Commands

The default syntax of a Registry section is oriented at the command syntax of other patch operations in opsi-script.

There exist the following commands:

- `OpenKey`
• Set
• Add
• Supp
• GetMultiSZFromFile
• SaveValueToFile
• DeleteVar
• DeleteKey
• ReconstructFrom
• Flushkey

In detail:

• OpenKey <registry key>
  opens the specified key for reading and (if the user has the necessary privileges) for writing. If the key does not exist it will be created.

The registry key is denoted by a registry path name. Under regular circumstances it starts with one of the "high keys" which build the top level of the registry tree data structure (above the "root"). These are: HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_CONFIG which may optionally be written as HKCR, HKCU, HKLM, HKU.

In opsi-script syntax of the registry path name, the elements of a path are separated by single backslashes.

All other commands operate on an opened registry key.

• Set <varname> = <value>
  sets the registry variable <varname> to value <value>. <varname> as well as <value> are strings and have to be enclosed in citations marks. A non-existing variable will be created. As default data typ normally REG_SZ is used. But if <value> contains a percent char (%) REG_EXPAND_SZ will be used instead.

The empty variable "" denotes the standard entry of a registry key.

If some registry variable shall be created or set where the data type should be explicitly given, we have to use the extended variant of the Set command:

• Set <varname> = <registry type>:<value>
  sets the registry variable <varname> to value <value> of type <registry type>. The following registry types are supported:

  REG_SZ
  (string)

  REG_EXPAND_SZ
  (a string containing substrings which the operating system shall expand e.g.)

  REG_DWORD
  (integer values; decimal or 0xhex)

  REG_BINARY
  (binary values usually given as two-digit hex numbers 00 01 02 .. 0F 10 ..)

  REG_MULTI_SZ
  (string value arrays, in opsi-script we have to use "|" as separator)

  An example for setting a REG_MULTI_SZ:

  ```
  set "myVariable" = REG_MULTI_SZ:"A|BC|de"
  ```
To construct a multistring we may put the strings as lines in a file and read it using \texttt{GetMultiSZFromFile} (cf. below).

Example for \texttt{set} with different registry data 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"
```

- \textbf{Add} \texttt{<varname> = <value>}

  resp.

  \textbf{Add} \texttt{<varname> = <registry type> \<value>}

  are analogous to the \texttt{Set} commands with the difference that entries are only added but values of existing variables not changed.

- \textbf{Supp} \texttt{<varname> \<list separator> \<supplement>}

  This command interprets the string value of variable \texttt{<varname>}, a list of values separated by \texttt{<list separator>} and adds the string \texttt{<supplement>} to this list (if it not already contained). If \texttt{<supplement>} contains the \texttt{<list separator>} it is split into single strings, and the procedure is applied to each single string.

  A typical use is adding entries to a path variable (which is defined in the registry).

  \textbf{Supp} keeps the original string variant (\texttt{REG_EXPAND_SZ} or \texttt{REG_SZ}).

Example:
The environment \texttt{Path} is determined by the value for the variable \texttt{Path} as defined inside the registry key

\texttt{\texttt{KEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment}}

To add some entries to the path definition we have to get access to this key via an OpenKey. Then we can apply e.g.

\texttt{\texttt{supp \"Path\" ; \"C:\utils;%JAVABIN\%\"}}

+ in order to supplement the path by \texttt{\"C:\utils\"} and \texttt{\"%JAVABIN\%\"}.

+ (Windows expands \texttt{\%JAVABIN\%} to the correct path name if \texttt{\%JAVABIN\%} exists as variable and the String is a \texttt{REG_EXPAND_SZ}.)

+ Whom read the old value of \texttt{Path} from the environment variable, write this value to the registry value - and are then able to work with the registry variable:

\texttt{\[Actions\]}
\texttt{DefVar $Path$}
\texttt{set $Path$ = EnvVar (\"Path\")}
\texttt{Registry\_PathPatch}
\texttt{\[Registry\_PathPatch\]}
\texttt{openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\control\Session Manager\Environment]} 
\texttt{set "Path"="$Path$"}
\texttt{supp "Path"; \"c:\orawin\bin\"}

\textbf{CAUTION}: The environment variable gets a changed value after a reboot or after a call of \texttt{UpdateEnvironment} see: \texttt{[UpdateEnvironment]}

- \textbf{GetMultiSZFromFile} \texttt{<varname> <filename>}

  reads the lines of a file and puts them together building a Multistring.

- \textbf{SaveValueToFile} \texttt{<varname> <filename>}

  exports the referred (String or MultiSZ) value as file \texttt{<filename> lines} (each String forming a line).

- \textbf{DeleteVar} \texttt{<Varname>}

  removes the entry with variable \texttt{<varname>} from the opened key.
• **DeleteKey** `<RegistryKey>`
  deletes the registry key recursively including all subkeys and contained variables. The registry key is defined as for `OpenKey`.

  Example:

  ```plaintext
  [Registry_KeyDelete]
  deletekey [HKCU\Environment\subkey1]
  ```

• **ReconstructFrom** `<filename>`
  (deprecated)

• **FlushKey**
  ensures that all entries of a key are saved on hard drive, not only in memory (is automatically done when closing a key, therefore in particular when a registry section is left).

### 10.13.4. Registry Sections to Patch `All NTUser.dat`

A Registry section called with parameter `/AllNTUserdats` is executed for every local user.

To this end, for all local users (as permanent storage for the registry branch `HKEY_Users`) the files `NTUser.dat` are searched one by one and temporarily loaded into a subkey of some registry branch. The commands of the registry section are executed for this subkey, then the subkey is unloaded. As result, the stored `NTUser.dat` is changed.

The mechanism does not work for a logged in user. For, his `NTUser.dat` is already in use, and the request to load it produces an error. To do the changes for him as well, the commands of the Registry additionally are executed on the `HKEY_Users` branch for the logged in user.

There is a `NTUser.dat` for Default User which serves as template for newly created users in the future. Therefore the patches are prepared for them as well.

The Registry section syntax remains unchanged. But the key paths are interpreted relatively. This means **leave away the main key**.

In the following example the registry entry for variable `FileTransferEnabled` is de facto set for all `HKEY_Users\XX\Software…` successive for all XX (all users) on the machine:

```plaintext
[Registry_AllUsers]
openkey [Software\ORL\WinVNC3]
set "FileTransferEnabled"=reg_dword:0x00000000
```

Since `opsi-script` version 4.11.2 you may give the root key `HKEY_CURRENT_USER` at the `openkey` command.

Example:

```plaintext
[Registry_AllUsers]
openkey [HKEY_CURRENT_USER\Software\ORL\WinVNC3]
set "FileTransferEnabled"=reg_dword:0x00000000
```

This has the following advantages:

• The syntax is easier to understand.

• The same registry section may be used with `/AllNTuserdats` and in a `userLoginScript`

### 10.13.5. Registry Sections in Regedit Format

If a Registry section is called with parameter `/regedit` the section is not expected in `opsi-script` standard format but in the format as produced by the Windows regedit tool. The export files generated by regedit have - not regarding the head line - ini file format. Example:
The sections denote registry keys to be opened. Each line describes some variable setting like the set command in a *opsi-script* registry section.

But, we cannot really have an internal *opsi-script* section that is constructed from another sections. Therefore Registry section with parameter `/regedit` can only be given as external section or by the function call `loadTextFile`, e.g.

```
registry "%scriptpath%/opsiorgkey.reg" /regedit
```

With Windows XP the registry editor `regedit` does not produce Regedit4-Format but a new format that is indicated by the head line

"Windows Registry Editor Version 5.00"

In this format, Windows offers some additional value types. But more important, the export file is now generated in Unicode. *opsi-script* sections processing is based on Delphi libraries which use 8 bit Strings. To work with a regedit 5 export the coding therefore has to be converted. This can be done manually, e.g. by a suitable editor. But we may also feed the original file to *opsi-script* using the String list function `loadUnicodeTextFile`. E.g., if `printerconnections.reg` be a unicode based export, we can call `regedit` in the following form which does the necessary code conversion on the fly:

```
registry loadUnicodeTextFile("%scriptpath%/opsiorgkey.reg") /regedit
```

A registry patch using regedit format can as well be executed “for all NT users” similarly as the common *opsi-script* registry section. That is, a path like `[HKEY_CURRENT_USER\Software\ORL\]` is to replaced by the relative `[Software\ORL]`.

### 10.13.6. Registry Sections in AddReg Format

A Registry section can be called with parameter `/addReg`. Then its syntax follows the principles of the `[AddReg]` sections in inf files as used e.g. for driver installations.

E.g.:

```
[Registry_ForAcroread]
HKCR,.fdf","","","AcroExch.FDFDoc"
HKCR,.pdf","","","AcroExch.Document"
```

### 10.14. OpsiServiceCall Sections [W/L/M]

This type of section allows to retrieve information – or set data – via the opsi service. There are three options for determining a connection to an opsi service:

- **Per default it is assumed that the script is executed in the standard opsi installation environment. I.e., we already have a connection to an opsi service and can use it.**

- **We set the url of the service to which we want to connect as a section parameter and supply as well the required username and password as section parameters.**

- **We demand an interactive login to the service (predefining only the service url and, optionally, the user name).**
Retrieved data may be returned as a String list and then used for scripting purposes.

10.14.1. Call Parameters

There is a standard webservice connection. This is established at the start of the opsi-script via the opsi-client-agent on the existing connection to the current opsi-server. If no call parameters are specified, the standard connection is applied. If it’s not specified, then the call fails. There are a number of call parameters that create a new connection. This new connection will be the default connection. That means, that subsequent calls without parameters use this connection until it is explicitly changed again, or the product script has been processed. A new product starts the original webservice connection again.

The call parameters that change the standard connection:

- `/interactive`
- `/serviceurl /username /password`
- `/opsiclientd`

Restore of the original connection:

With the `opsiServiceCall` section with a parameter call `/preloginservice` the standard connection will be restore to the previous value. Alternatively one can also make a call to the statement without an actual existing section:

`opsiServiceCall /preloginservice`

The parameter call:

The call parameters determine which opsi service will be addressed and set the connection parameters if needed.

Connection parameters can be defined via

- `/serviceurl <url to the opsi web service>`
- `/username <web service user name>`
- `/password <web service user password>`

If these parameters are defined (or at least one of them), an attempt is made to connect to the mentioned service URL and, if successful, then this will be the default connection.

The additional option

- `/interactive`

raises an interactive connect. The user will be asked for confirming the connection data and supplying the password. Of course, this option cannot be used in scripts which shall be executed fully automatically.

If no connection parameters are supplied `opsi-script` assumes that an existing connection shall be reused.

If no connection parameters are given and the interactive option is not specified (neither at this call nor at a call earlier in the script) it is assumed that we are in a standard opsi boot process and, already having a connection to an opsi service, we try to address it.

- `/preloginservice`
  In the case that we had a connection to a secondary opsi service we may (re)set the connection to the standard opsi service via the option

- `/opsiclientd` //since 4.11.2.1
  calls the localhost opsiclientd

- `/opsiclientd-once` //since 4.11.6.11
  Calls the webservice from the local opsiclientd and sets back after the call, the standard connection once again to the original value.
**10.14.2. Section Format**

An `opsiServiceCall`, which uses an existing connection to an opsi-service, is defined by its method name and a list of parameters.

Both are defined in the section body. It has format

```
"timeout":<seconds>,
"method":<method name>,
"params":[
  ...
]
```

The line "timeout":<seconds> is optional.

It is only needed by calling methods that run very long (> 90 seconds). It is available since 4.12.4.35.

`params` is a (possibly empty) list of strings (comma-separated). Use the parameters as required by the specified method.

E.g. we may have a section call where the required methodname and the (empty) list of parameters is set:

```
[opsiservicecall_clientIdsList]
"timeout":50,
"method":"getClientIds_list",
"params":[]
```

The section call produces the list of names (IDs) of all local opsi clients. If the list shall be exploited for other than test purposes the section call can be used in a string list expression: E.g.:

```
DefStringList $result$
Set $result$=getReturnListFromSection("opsiservicecall_clientIdsList")
```

The usage of `GetReturnListFromSection` is documented in the string list function chapter of this manual (see Section 9.4.5, “Producing String Lists from opsi-script Sections [W/L/M]”).

A hash – in this case a string list, where each item is a pair name=value – is produced by the following opsi service call:

```
[opsiservicecall_hostHash]
"method": "getHost_hash",
"params": [ "pcbon8.uib.local"
]
```

### Object oriented Methods

Dealing with JSON objects from the web service requires a basic understanding of JSON, the opsi objects and the JSON-related methods in opsi-script. See also: Opsi-manual: Chapter: “Web service / API methods since opsi 4.0”:


In the code shown below, you can get objects from the service. In this example, all productOnClient objects that belong to the current computer will be retrieved (`%opsiserviceUser%` are in the service context of the FGDN of the client) and are localboot products, by which the action request is set to `setup`.

```
DefStringlist $resultlist$
set $resultlist$ = getReturnListFromSection("opsiServiceCall_get_productOnClient_setup_objects")
```

The result is a JSON Array String which is in the first line of `$resultlist$`.

You can also restore (changed) objects. The following example clarifies the principle: The string variable `$ArrayStr$` must contain a valid JSON array.

---

ub
10.14.3. Examples

For further examples watch the product opsi-script-test and there especially $Flag_winst_opsiServiceCall$ = "on"

10.15. ExecPython Sections [W/L/M]

ExecPython sections are basically Shell-Sections (like DosInAnIcon) which call the – on the system installed – python script interpreter. It takes the section content as python script, and the section call parameter as parameters for the script.

Example:

The following example demonstrates a execPython call with a list of parameters for that are printed by the python commands.

The call may look like

```python
execpython_hello -a "option a" -b "option b" "there we are"
```

```python
[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"
```

The print command output will be caught and written to the log file. So we get in the log

```text
output:
--------
-a
option a
-b
option b
there we are
hello
```

Observe that the loglevel must be set at least to info (that is 1) if these outputs shall really find their way to the log file.

10.15.1. Interweaving a Python Script with the opsi-script Script

An execPython section is integrated with the surrounding opsi-script script by four kinds of shared data:

- A parameter list is transferred to the python script.
- Everything that is printed by the python script is written into the opsi-script log.
- The opsi-script script substitution mechanism for constants and variables when entering a section does its expected work for the execPython section.
- The output of an execPython section can be caught into a stringlist and then used in the ongoing opsi-script script.

An example for the first two ways of interweaving the python script with the opsi-script script is already given above. We extend it to retrieve the values of some opsi-script constants or variables.
Of course, the $loglevel$ variable has to be set beforehand in the Actions section:

```
DefVar $loglevel$
set $loglevel$ = getLoglevel
```

Finally, in order to being able to use of some results of the section output, we produce it into a stringlist variable by calling the execPython section in the following way:

```
DefStringList pythonresult
Set pythonResult = GetOutStreamFromSection('execpython_hello -a "opt a"')
```

### 10.15.2. Examples

For further examples watch the product `opsi-script-test` and there especially $Flag_compare_to_python$ = "on"

### 10.16. ExecWith Sections [W/L/M]

*ExecWith* sections are more general than *ExecPython* or *DosBatch* sections: Which program interprets the section lines given is determined by a parameter of the section call.

E.g. if we have some call

```
execPython_hello -a "hello" -b "world"
```

where

- `-a "hello"` - `-b "world"`

are parameters that are passed to the python script we get the following completely equivalent ExecWith call:

```
execWith_hello "python" PASS -a "hello" -b "world" WINST /EscapeStrings
```

The option `/EscapeStrings` is automatically used in an ExecPython section and means that backslashes in String variables and constants are duplicated before interpretation by the the called program.

#### 10.16.1. Calling parameters (Modifier)

In general, we have the call syntax:

```
ExecWith_SECTION PROGRAM PROGRAMPARAS pass PASSPARAS winst WINSTOPTS
```

Each of the expressions *PROGRAM, PROGRAMPARAS, PASSPARAS, WINSTOPTS* may be an arbitrary String expression, or just a String constant (without citation marks).

The key words PASS and WINST may be missing if the respective parts do not exist.
The following *opsi-script*-options are available:

- **/EscapeStrings**
- **/LetThemGo**
- **/32Bit** This is the default. The interpreter path is assumed to be a 32 bit path. Example: `c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe` calls (also when running on a 64 bit system) the 32 bit `powershell.exe`.

- **/64Bit** The interpreter path is assumed to be a 64 bit path. Example: `c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe` calls (on a 64 bit system) the 64 bit `powershell.exe`.

- **/SysNative** The given interpreter path is assigned accoring to the OS architecture. Example: `c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe` calls on a 64 bit system the 64 bit `powershell.exe` and running on a 32bit system the 32 bit `powershell.exe`.

*/encoding <encoding> / since 4.12.4.17 [W/L/M]*

By default the content of the section will be stored to a temporary file in system encoding. This is in the most cases the best choice. If you really knows what you are doing, you may change the encoding of the stored temporary file:

You can add an encoding parameter after the WINST keyword. Example:

```plaintext
ExecWith_open "C:\myProgram.exe" WINST /encoding "utf8"
```

For allowed encodings see *opsi-script* `encoding`

Like with ExecPython sections, the output of an ExecWith section may be captured into a String list via the function `getOutStreamFromSection`.

The first one declares that the backslash in *opsi-script* variables and constants is C-like escaped. The second one has the effect (as for `winBatch` calls) that the called program starts its work in new thread while *opsi-script* is continuing to interpret its script.

Since Version 4.11.3.5, if the interpreter path contains `powershell.exe`, the temporary file is saved with the extension `.ps1`.

### 10.16.2. More Examples

The following call is meant to refer to a section which is an autoit3 script that waits for some upcoming window (therefore the option /letThemGo is used) in order to close it:

```plaintext
ExecWith_close "%SCRIPTPATH%\autoit3.exe" WINST /letThemGo
```

A simple

```plaintext
ExecWith_edit_me "notepad.exe" WINST /letThemGo
```

calls notepad and opens the section lines in it (but without any line that is starting with a semicolon since *opsi-script* regards such lines as comments and eliminates them before handle).

The following example call the 64Bit version of the `powershell.exe`.

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

**Note** **Note on powershell** For Powershell the script execution is disabled by default. So you have to enable it before you can use Execwith with powershell. To achieve that there are two options: Either call the section using `ExecWith_name "powershell.exe" -ExecutionPolicy Bypass` to temporarily allow execution or enable it permanently using `DosInAnIcon` beforehand.
Since 4.12.4.35: If the ExecutionPolicy AllSigned is detected, the powershell will be modified from calling the temporary script by -File to -Command Get-Content -Path <tempfilename> | Out-String | Invoke-Expression. In this mode the ExecutionPolicy is ignored. In this mode any given PASSPARAS are also ignored.

Example

```powershell
DosInAnIcon_setpolicy
ExecWith_powershell powershell.exe
set $exitcode$ = getLastExitcode
if not ($exitcode$ = "0")
    comment "powershell script failed"
endif

[DosInAnIcon_setpolicy]
powershell.exe set-executionpolicy RemoteSigned ...
exit \ERRORLEVEL\)

[ExecWith_powershell]
echo "powershell opsi-script-test"
if ($?) {Exit(0)}
else {Exit(1)}
```

For further examples watch the product opsi-script-test and there especially $Flag_autoit3_test$ = "on"

### 10.17. LDAPsearch Sections [WLM]

A LDAPsearch section defines a search request to a LDAP directory, executes it and receives (and possibly caches) the response.

Before dwelling on the opsi-script commands we do some explanations.

In subsection we give an example of the most probable usage of a LDAPsearch. The following subsections describe the syntax


LDAP, the "Lightweight Directory Access Protocol", is, as the name indicates, a defined way of communication to a directory. This directory is (or may be) hierarchically organized. That is, the directory is a hierarchical data base, or a tree of content.

A LDAP service implements the protocol. A directory that can be accessed via a LDAP service is called a LDAP directory.

For instance, let's have a look at some part of the LDAP directory tree of an opsi server with LDAP backend (as shown by the Open Source tool JXplorer):
Figure 9. View of some part of an opsi LDAP tree

A **LDAP search request** is a query to a LDAP directory via a LDAP service. The response returns some content from the directory.

Basically the search request has to describe the path in the directory tree which leads to the interesting piece of information. The path is the **distinguished name** (dn), composed of the names of the nodes (the "relative distinguished names"), which build the path, for instance:

```
local/uib/opsi/generalConfigs/bonifax.uib.local
```

Since each node is conceived as an instance of some structural object class, the path description is usually given in the following form: with indication of the classes (and starting with the last path element):

```
cn=bonifax.uib.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
```

The path in a query is not necessarily "complete", and not leading to a unique leaf of the tree. On the contrary, partial paths are common.

But even if the path descends to a unique leaf, the leaf may contain several values. Each node of the tree has one or more classes as attribute types. To each one or may values may be associated.

For a given query path, we therefore may be interested

1. in the node set whose elements – the so called LDAP objects – match the given path,
2. the set of attributes that belong the nodes,
3. and the values that are associated to all of them.

Obviously, handling the amount of possibly returned response information is the main challenge when dealing with LDAP searches.

The following section shows the documentation of a LDAP search roughly corresponding to the screenshot above as executed by **opsi-script**.

**Example**

Using the **opsi-script** section **ldapsearch_generalConfigs**: 

```
```
we will get a answer like this:

```
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
secuntilconnectiontimeout=280
opsiclientd.global.log_level=
```

There are several opsi-script options to manage and reduce the complexity of the evaluation of such responses.

### 10.17.2. LDAPsearch Call Parameters

Two kinds of LDAPsearch parameters,

- cache options
- output options

are defined for the call of LDAPsearch section.

The cache options are:

- `/cache`
- `/cached`
- `/free`
- (no cache option)

If there is no cache option specified, the response of the LDAP search request is not saved for further usages.

By the `/cache` option, the response is cached for further evaluations, the `/cached` option refers to the last cached response which is reused instead of starting a new search, the `/free` option clears the cache explicitly (may only be useful for searches with extreme large responses).

The output options are:

- `/objects`
- `/attributes`
- `/values`
The output options determine the String list that is produced when a LDAP search section is called via getReturnlistFromSection:

- If no output option is specified the returned list is the complete LDAP response.
- The options objects, attributes and values restrict the output to object, attribute or value lines of the LDAP response respectively.

Observe that in the produced lists the object an attribute belongs to is only identifiable if only one object is returned in the object list, and likewise the object and the attribute to which a value is subsumed are only identifiable if there is only attribute remaining in the attributes list.

Such the proceeding is, that the LDAP search is specified up to that degree, that at most one object and one attribute is returned. This can be checked by a count call on the objects and the attributes return list. Then any value found belongs to the DN and the attribute specified.

The repeated utilization of the same LDAP response can be done without relevant time costs by using the cache/cached options.

**10.17.3. How to Narrow the Search**

An example may show how we can narrow the search to pin down a specific result from a LDAP directory.

We start with the call of `ldapsearch_generalConfigs` as above, only adding the cache parameter.

```bash
ldapsearch_generalconfigs /cache
```

executes the query and caches the response for further utilization.

Then, the call

```bash
getReturnlistFromSection("ldapsearch_generalconfigs /cached /objects")
```

produces the list

```

cn=generalconfigs,cn=opsi,dc=uib,dc=local

cn=bonifax.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local

cn=bonifax.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
```

If we narrow the tree selection by

```bash
[ldapsearch_generalConfigs]
targethost: bonifax

dn: cn=bonifax.ubi.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
```

and start again, then in the objects list, we indeed retain just

```

cn=bonifax.ubi.local,cn=generalconfig,cn=opsi,dc=uib,dc=local
```

The corresponding attributes list contains three elements:

```
objectclass

cn

cn=opsikeyvaluepair
```

In order to get the values associated to a single attribute we have to confine the query once more:

```bash
[ldapsearch_generalConfigs]
targethost: bonifax

dn: cn=bonifax.ubi.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local

attribute: cn=opsikeyvaluepair
```

The result now produced is an attributes list containing only one element. The corresponding values list looks like
There are no LDAP means to reduce this result furthermore!

(But the `opsi-script` function `getValue(key, list)` (cf. Simple String Values generated from String Lists or Files) may help in this case: E.g. `getValue("secsuntilconnectiontimeout", list)` would produce the requested number).

By the function `count(list)` we can check if we succeeded with the narrowing of the search request. In most circumstances, we would like that its result be “1”.

### 10.17.4. LDAPsearch Section Syntax

A LDAPsearch section comprises the specifications:

- **targethost**: The server hosting the LDAP directory (service) must be named.
- **user**: user name to be applied. Since 4.11.3.5
- **password**: user password to be applied. Since 4.11.3.5
- **targetport**: If the port of the LDAP service is not the default (389), it can be declared at this place. If the specification is missing, the default port is used. Since 4.12.4.29: Using the port 636 turns also using connection via SSL on.
- **dn**: Here, the distinguished name, the "search path", for the search request can be given.
- **typesonly**: Default "false", that is, values are retrieved.
- **filter**: A filter for LDAP search has a LDAP specific syntax that is not checked by `opsi-script`. Default is "(objectclass=*)"
- **attributes**: A comma separated list of attribute names may be given. The default is to take any attribute.

### 10.17.5. Examples

A short and rather realistic example shall end this section:

$founditems$ be a `StringList` variable and $SopsiClient$ a `String` variable. The call of `getReturnlistFromSection` fetches the results of the section `ldapsearch_hosts`. The following code fragment returns the unique result for $SopsiDescription$ if it exists. It reports an error if the search produces an unexpected result:
Example with user / password

```plaintext
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 $LdapFilter$ = "(a(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: $LdapFilter$
```

For further examples watch the product `opsi-script-test` and there especially `$Flag_winst_ldap_search$ = "on"`
Chapter 11. 64 Bit Support on Windows [W]

The opsi-script is a 32 bit program. In order to make it easy for 32 bit programs to run on 64 bit systems there are special 32 bit areas in the registry as well in the file system. Some accesses from 32 bit programs will be redirected to these special areas to avoid access to areas that reserved for 64 bit programs.

A access to \c:\windows\system32 will be redirected to \c:\windows\syswow64

But a access to \c:\program files will be not redirected to \c:\program files (x86)

A registry access to [HKLM\software\opsi.org] will be redirected to [HKLM\software\wow6432node\opsi.org]

Therefore opsi-script installs as 32 bit program scripts, that run on 32 bit system fine, on 64 bit system correct without any change.

For the installation of 64 bit programs some constants like %ProgramFilesDir% returns the wrong values. Therefore we have since opsi-script 4.10.8 some new features:

Normally you may (and should) tell explicit to which place you want to write or from where you want to read. Here we have three variants:

32

explicit 32 bit

64

explicit 64 bit; if not on a 64 bi system like sysnative

SysNative

according to the architecture on which the script runs

Following this idea, we have some additional constants:

Table 5. Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>32 Bit</th>
<th>64 Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ProgramFilesDir%</td>
<td>c:\program files</td>
<td>c:\program files (x86)</td>
</tr>
<tr>
<td>%ProgramFiles32Dir%</td>
<td>c:\program files</td>
<td>c:\program files (x86)</td>
</tr>
<tr>
<td>%ProgramFiles64Dir%</td>
<td>c:\program files</td>
<td>c:\program files</td>
</tr>
<tr>
<td>%ProgramFilesSysnativeDir%</td>
<td>c:\program files</td>
<td>c:\program files</td>
</tr>
</tbody>
</table>

%ProgramFilesDir%

you should avoid this in future…

%ProgramFiles32Dir%

should be used in the context of installing 32 bit Software.

%ProgramFiles64Dir%

should be used in the context of installing 64 bit Software.

%ProgramFilesSysnativeDir%

should be used if you need architecture specific information

For a reading access to the different areas of registry and filesystem we have now the following new functions:

- GetRegistrystringValue32
- GetRegistrystringValue64
The following functions have the possibility to control the access mode by a parameter (the default is here `sysnative`):

- `getRegistryValue`
- `RegKeyExists`
- `RegVarExists`
- `powershellCall`

A simple call to Registry-section results in writing to the 32 bit registry regions. Also a simple call to Files-section results in writing to the 32 bit file system regions.

For `Registry`, `Files` and `Winbatch` sections we have now the additional calling options:

- `/32Bit`  
  This is the default. Any access will be redirected to the 32 bit regions.

- `/64Bit`  
  Any access will be redirected to the 64 bit regions. If there are no 64 bit regions the architecture specific regions will be used.

- `/SysNative`  
  Any access will be redirected to the architecture specific regions

For `DosBatch`, `DosInAnIcon` (`ShellBatch`, `ShellInAnIcon`) and `Execwith` you have to keep in mind that any modifiers has to separated by the keyword `winst`.

Example:

```
DosInAnIcon_do_64bit_stuff winst /64Bit
```

In addition to these `opsi-script` functions, we copy at the installation of the opsi-client agent the (64 bit) file `c:\windows\system32\cmd.exe` to `c:\windows\cmd64.exe`. Using this `cmd64.exe` with `ExecWith` sections you may call any 64 bit operations on the command line.

Examples:

File handling:

```bash
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
else
```
Registry Handling:

```powershell
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
    endif
    else
        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
    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
    Registry_hkcu_opsi_org_test /AllNtUserDats /64Bit
else
    set $regWriteValue$ = "32"
    Registry_hkcu_opsi_org_test /AllNtUserDats
    Registry_hkcu_opsi_org_test /AllNtUserDats /64Bit
endif
```
Chapter 12. Cook Book

This chapter contains a growing collection of examples showing real world problems that can be mastered by simple or sophisticated pieces of *opsi-script* scripting.

12.1. Delete a File in all Subdirectories

Since *opsi-script* 4.2 there is an easy solution for this task: To remove a file alt.txt from all subdirectories of the user profile directory the following `Files` call can be used:

```bash
files_delete_Alt /AllUserProfiles
```

```bash
files_delete_Alt
delete "%UserProfileDir%\alt.txt"
```

Nevertheless we document a workaround which could be used in older *opsi-script* versions. It demonstrates some techniques which may be helpful for other purposes.

The following ingredients are needed:

- A `DosInAnIcon` section which produces a list of all directory names.
- A `Files` section which deletes the file alt.txt in some directory.
- A `String list processing` that puts the parts together.

The complete script should look like:

```bash
[Actions]
; variable for file name
DefVar $deleteFile$ = "alt.txt"

; String list declarations
DefStringList list0
DefStringList list1

; capture the lines produced by the `dos dir` command
Set list0 = getOutStreamFromSection ('dosbatch_profiledir')

; Loop through the lines. Call a `Files` section for each line.
for $x$ in list0 do files_delete_x

; Here are the two special sections
[dosbatch_profiledir]
@dir "%ProfileDir%\" /b

[files_delete_x]
delete "%ProfileDir\%x\%deleteFile%"
```

12.2. Check if a specific service is running

If we want to check if a specific service (exemplified with "opsiclientd") is running, and, e.g., if it is not running, start it, we may use the following script.

In order to get the list of running services we launch the command

```bash
net start
```

in a `DosBatch` section, capturing its output in `list0`. We trim the list, and iterate through its elements, thus seeing if the specified service is in it. If not, we do something for it.
12.3. Script for installations in the context of a local user

Sometimes it is necessary to run an installation script as a logged in local user and not in the context of the opsi service. For example, there are installations that require a user context or use services that are only started after a user login. MSI installations that require a local user can sometimes be configured by the option `ALLUSERS=1` to proceed without such a user:

```powershell
msiexec %SCRIPTPATH%\files\myproduct.msi /qb ALLUSERS=1 /l* $LOG_LOCATION$ /i
```

12.3.1. opsi-template-with-userlogin

Another solution for this problem is to create a temporary local user and run the installation while it is logged in. For this scenario we offer the product `opsi-template-with-userlogin`, which supersedes the product `opsi-template-with-admin`.

Always use the latest version of `opsi-template-with-userlogin`!

Customizing the product

To customize the template to fit your needs it is recommended to create a new product, based on `opsi-template-with-userlogin`:

```powershell
opsi-package-manager -i --new-product-id myproduct opsi-template-with-userlogin_4.x.x.x-opsi
```
Workflow

During the installation the following steps are processed:

- Backup of the following values:
  - Current Auto Logon settings.
  - Last logged in user.
  - User Account Control settings.
  - Host parameter opsiclientd.event_software_on_demand.shutdown_warning_time.
- Temporarily setting the host parameter opsiclientd.event_software_on_demand.shutdown_warning_time to 0, to avoid unnecessary delays.
- Generation of a random password for the opsiSetupUser.
- Creation of the local user opsiSetupUser.
- Setup of the Auto Logon function for the user opsiSetupUser.
- Creation of a Scheduled Tasks for the installation in the Task Scheduler.
- Copying the installation files to the client. (Depending on the settings of the Product Property execution_method)
- Reboot of the client so that the Auto Logon settings take effect.
- Automatic login of the opsiSetupUser.
- Running the installation via the Scheduled Task. The task starts with one minute delay in order to give all the services enough time to start.
- Reboot of the client after the installation finishes.
- Cleanup and restore of the formerly backed up values.
  - Deletion of the opsiSetupUser including the user profile and all registry entries.
  - Deletion of all local files.
  - Restoration of the former values for Auto Logon, last logged on user and User Account Control.
  - Restoration of the former value of the host parameter opsiclientd.event_software_on_demand.shutdown_warning_time.

Product Properties

The behaviour of the product can be customized via the following product properties:

**debug**

- False (Default)
  - Disables mouse and keyboard input during the Auto Logon to prevent user interaction. The password of the opsiSetupUser is not plainly visible in the log file.
- True
  - Keyboard and mouse input remain enabled during the Auto Logon. The password of the opsiSetupUser is written in plain text in the log file.

**execution_method**

- event_starter_local_files
  - The installation is triggered via the opsiclientd_event_starter_asInvoker.exe during the Auto Logon, which contacts the server and triggers an on_demand event.
  - The installation runs in the context of the user System.
  - The opsiSetupUser is created without admin rights.
  - The installation files are copied locally to the client.
- The installation is triggered via the opsiclientd_event_starter_asInvoker.exe during the Auto Logon, which contacts the server and
triggers an on_demand event.
- The installation runs in the context of the user System.
- The opsiSetupUser is created without admin rights.
- The installation files remain on the opsi_depot share.

- The installation during the Auto Logon is run by the locally installed opsi-script.
- The installation runs in the context of the user opsiSetupUser.
- The opsiSetupUser is created with admin rights.
- The installation files are copied locally to the client.

- If the client is using the WAN/VPN mode (determined automatically) this Product Property is ignored and the installation runs with the
following settings:
  - The installation during the Auto Logon is run by the locally installed opsi-script.
  - The installation runs in the context of the user opsiSetupUser.
  - The opsiSetupUser is created with admin rights.
  - The installation files from the local cache are used.

uninstall_before_install

- False (Default)
  - No uninstallation takes place prior to the installation.
- True
  - Checks if a the software is already installed prior to the installation. If that is the case the software will be uninstalled before the
installation starts.

Structure of the product

The product is divided into a main script that prepares the Auto Logon and the installation, and an installation script that is triggered during
the Auto Logon of the local user.

Main script

For the sake of readability the main script is split into the following files:

- declarations.opsiinc (Contains the definition of all the used variables)
- sections.opsiinc (Contains all the sections used in the main script)
- setup.opsiscript

The only changes that need to be made to the main script are the settings for the required available free space and the parameters for the
generation of the random passwort used for the opsiSetupUser. These need to be made in the file declarations.opsiinc:
Installation script

The installation script is split into multiple files as well:

- declarations-local.opsiinc (Contains the definition of all the used variables)
- sections-local.opsiinc (Contains all the sections used in the installation script)
- setup-local.opsiinc
- delsub-local.opsiinc
- uninstall-local.opsiscript

Adding the installation files

Open the directory of your product in the servers depot and copy the installation files into the folder `localsetup\files`. The files `Testfolder1` and `Testfile1.txt` can safely be deleted.

Customizing the variables

Customize the variables in `localsetup\declarations-local.opsiinc` to fit your needs:

Customizing setup-local.opsiinc

The file `setup-local.opsiinc` contains the handling of the installation and the license management, as well as examples for the copying of files and folders and the creation of registry entries and desktop shortcuts. The example sections are commented out by default. These can be
safely deleted, remain commented out or used, depending on your needs.

Customizing sections-local.opsiinc

This file contains all the sections that are needed for the installation. You need to uncomment the appropriate function to evaluate the exit codes of your installer type in the section [Sub_Check_ExitCode]. The exit codes of the following installer types can be evaluated:

- Inno Setup
- InstallShield
- MSI
- Nullsoft Scriptable Install System (NSIS)

The installer type can be determined using the tool `opsi-setup-detector`.

In this example the function `isMsiExitcodeFatal` is used:

```plaintext
[Sub_Check_ExitCode]
Set $ExitCode$ = getlastexitcode
;if stringtobool(isInnoExitcodeFatal($ExitCode$, "true", $ErrorString$ ))
  Set $ErrorFlag$ = $ErrorString$
  Registry_Save_Fatal_Flag /32Bit
  ExitWindows /ImmediateReboot
  else
    Comment $ErrorString$
endif
```

The sections `Winbatch_Install` and `Winbatch_Uninstall` contain commented out examples for the installation and deinstallation commands used by the different installer types. Uncomment and customize the appropriate commands for the installer type that your software uses.

```plaintext
[Winbatch_Install]
; Choose one of the following examples as basis for your installation
; You can use the variable $LicenseKey$ to pass a license key to the installer

;======== Inno Setup =======
;"%ScriptPath%\localsetup\files\setup.exe" /sp- /silent /norestart

;======== InstallShield =======
;Create an setup.iss answer file by running: setup.exe /r /f1"c:\setup.iss"
;r"%ScriptPath%\localsetup\files\setup.exe" /a /msi /f1"%ScriptPath%\localsetup\files\setup.iss" /f2"%LogDir%\$ProductId$.install_log.txt"

;======== MSI package =======
;msixex -i "%ScriptPath%\localsetup\files\setup.msi" /gb! /i! "%LogDir%\$ProductId$.install_log.txt" ALLUSERS=1 REBOOT=ReallySuppress

;======== Nullsoft Scriptable Install System (NSIS) =======
;r"%ScriptPath%\localsetup\files\setup.exe" /S <additional_parameters>

[Winbatch_Uninstall]
; Choose one of the following examples as basis for your uninstallation

;======== Inno Setup =======
;"$Uninstaller$" /silent /norestart

;======== InstallShield =======
;Create an uninstall.iss answer file by running: setup.exe /uninst /r /f1"c:\uninstall.iss"
;r"%ScriptPath%\localsetup\files\setup.exe" /uninst /a /f1"%ScriptPath%\localsetup\files\uninstall.iss" /f2"%LogDir%\$ProductId$.uninstall_log.txt"

;======== MSI =======
;msixex -x $MsiId$ /gb! /i! "%LogDir%\$ProductId$.uninstall_log.txt" REBOOT=ReallySuppress

;======== Nullsoft Scriptable Install System (NSIS) =======
;"$Uninstaller$" /S
```

Customizing delsub-local.opsiinc

The handling of the uninstallation consists of either looking for an already installed executable, or a present MSI GUID in the registry. Uncomment the appropriate line for your installer type and comment out the other line. In the following example the line for MSI is uncommented:
Comment "Searching for already installed version"
; if FileExists($InstalledExecutable$)
if NOT(GetRegistryStringValue("\[HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" + $MsiId$ + "]\ DisplayName") = "")
Comment "Starting the uninstallation"
Winbatch_Uninstall /SysNative
Sub_Check_ExitCode

Comment "License handling"
; if NOT($LicenseRequired$ = "false")
Comment "Licensing required, free license used"
Sub_Free_License
endif
; Comment "Deleting files"
; Files_Delete /SysNative
; Comment "Deleting registry entries"
; Registry_Delete /SysNative
; Comment "Deleting links"
; LinkFolder_Delete
endif

The file delsub-local.opsiinc contains the handling of the uninstallation and the license management, as well as examples for the deletion of files and folders, registry entries and desktop shortcuts. The example sections are commented out by default. These can be safely deleted, remain commented out or used, depending on your needs.

The uninstallation does not run in the context of the logged in local user, since this is usually not required.

Error handling

If you customize the scripts you need to make sure not to use the function isFatalError! The function isFatalError cancels the execution of the script immediately, which means that the cleanup phase re-enables keyboard and mouse inputs, restores the former settings and removes the opsiSetupUser will never be executed! This means the installation will stop with the logged in opsiSetupUser and it leads to an infinite Auto Logon loop after each reboot. To avoid this use the following code for the handling of errors. This stores the error message in the variable $ErrorFlag$, which will be saved in the registry. After that the client will be restarted via ExitWindows /ImmediateReboot immediately. After the reboot the cleanup phase will be executed and the value stored in the variable $ErrorFlag$ will be evaluated.

Set $ErrorFlag" = "Installation not successful"
Registry_Save_Fatal_Flag /32Bit
ExitWindows /ImmediateReboot

12.4. XML File Patching: Setting Template Path for OpenOffice.org 2

Setting the template path can be done by the following script extracts
12.5. Patching a XML configuration file for a MsSql application:
An example with misleadingly named attributes

The file which is to be patched has e.g. the following form; the values of DataSource and InitialCatalog will be filled using the variables $source$ and $catalog$.

```xml
<?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>
```

Then the following XMLPatch section can be used:
12.6. Retrieving Values From a XML File

As treated in Section 12.4,”XML File Patching: Setting Template Path for OpenOffice.org 2”, *opsi-script* can evaluate and modify XML files.

An example shall demonstrate how a value can be retrieved from a XML file. We assume that the following XML file is:

```xml
<?xml version="1.0" encoding="utf-16" ?>
xsi:schemaLocation="Collector.xsd" UtcDate="04/06/2006 12:28:17" LogId="(6930A32-766A-4F30-876C-611DEE955C2C)" 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="5" Minor="0"/>
  </SystemList>
  <SoftwareList>
    <Application Name="Windows XP-Hotfix - KB873333" ComponentType="Hotfix" EvidenceId="256" RootDirPath="C:\WINDOWS\uninstall\KB873333\spuninst" OsComponent="true" Vendor="Microsoft Corporation" Crc32="0x4235b909">
      <Evidence>
        <AddRemoveProgram DisplayName="Windows XP-Hotfix - KB873333" CompanyName="Microsoft Corporation" Path="C:\WINDOWS\uninstall\KB873333\spuninst" RegistryPath="HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KB873333" UninstallString="C:\WINDOWS\Uninstall\KB873333\spuninst\spuninst.exe" OsComponent="true" UniqueId="256"/>
      </Evidence>
    </Application>
    <Application Name="Windows XP-Hotfix - KB873339" ComponentType="Hotfix" EvidenceId="257" RootDirPath="C:\WINDOWS\uninstall\KB873339\spuninst" OsComponent="true" Vendor="Microsoft Corporation" Crc32="0x855c39c9">
      <Evidence>
        <AddRemoveProgram DisplayName="Windows XP-Hotfix - KB873339" CompanyName="Microsoft Corporation" Path="C:\WINDOWS\uninstall\KB873339\spuninst" RegistryPath="HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KB873339" UninstallString="C:\WINDOWS\Uninstall\KB873339\spuninst\spuninst.exe" OsComponent="true" UniqueId="257"/>
      </Evidence>
    </Application>
  </SoftwareList>
</Collector>
```

To read the elements and get the values of all „Application” nodes we may use these extracts of code:
We encapsulate the retrieved Strings by setting their values as a whole into a variable via an EscapeString call. Since the loop variable %line% is not a common variable but behaves like a constant all special characters in it (as < > $ % “ ‘ ) may cause difficulties.

12.7. Inserting a Name Space Definition Into a XML File

The opsici-script XMLPatch section requires fully declared XML name spaces (as is postulated in the XML RFC). But there are XML configuration files which do not declare „obvious“ elements (and the interpreting programs insist that the file looks this way). Especially patching the lots of XML/XCU configuration files of OpenOffice.org proved to be a hard job. For solving this task, A. Pohl (many thanks!) the functions XMLaddNamespace and XMLremoveNamespace. Its usage is demonstrated by the following example:

```
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
```

Please observe that the XML file must be formatted such that the element tags do not contain line breaks.

12.8. Finds out if a script is currently running in the context of a particular event

The opsici-client determines and knows which event is currently active. opsici-script can be used by means of an opsiservicecall And thus connect with the opsici-client querying the corresponding events:
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
Chapter 13. Special Error Messages

• No Connection with the opsi Service
  The opsi-script logs: "… cannot connect to service".

The information which is shown additionally may give a hint to the problem:

Socket-Fehler #10061, Connection refused
  Perhaps the opsi service does not run.

Socket-Fehler #10065, No route to host
  No network connection to server

HTTP/1.1. 401 Unauthorized
  The service responds but the user/password combination is not accepted.
Chapter 14. opsi documentation generator

14.1. Introduction

The opsi-doc-generator is a program to create documentation in asciidoc format from the following sources:

- opsi-script library files (opsi-script)
- opsi webservice interface definition files (python) (implementation is in progress)

Why asciidoc as output?

- asciidoc is the standard format for all opsi documentation
- asciidoc is a base format to create documents in different formats like html, pdf, epub, docbook, ...

The documentation is created from the information, that is extracted from the source code. From the source code opsi-doc-generator knows the definition of opsi-script defined functions and can get the information from there.

There may be special markers in comments 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 opsiscript 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 program 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 @ char and the marker identifier string.

Every allowed marker can occur never, once or multiple times on a level. If a marker occurs multiple times, all lines with this marker are concatenated.

After a marker one or more space chars have to be used before the information starts.

**opsi-script markers allowed on the file level**

- ;@author
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 on
- ;@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_<praram name> Description of the parameter <praram name>
- ;@ParamAdvice_<praram name> Advice for the parameter <praram name>. That may be for example used for restrictions of valid values.

14.4. opsi-doc-generator examples

opsi-script markers on the file level

```plaintext
;@author          detlef oertel
;@email          d.oertel@uib.de
;@date           17.4.2018
;@copyright      AGPLv3
;@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

```plaintext
@Author      detlef oertel
@Date        17.5.2018
@Description Sets for the given list of opsi productIds the action request
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)

@ParamDesc   $productlist$     List of opsi product Ids
@ParamAdvice $productlist$

@example [actions]
@example DefineStringList $productlist$
@example set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
@example if not(stringtobool(setProductsToSetup($productlist$))):
@example     comment "call of setProductsToSetup failed"
@example endif
```
Chapter 15. opsi-script Tutorial (1.0.0)

15.1. Introduction

This tutorial should help you to learn some advanced features (e.g. string lists) of the opsi-script script language.

Before we start some hints:

- you should always use opsi script constants if they applicable. For example use %system% instead of c:\windows\system32.
- You should use the opsi-script manuals for further description of the mentioned script commands:
  - opsi-script manual
  - opsi-script reference card
- You should use the opsi product opsi-script-test as a running reference script which is calling (nearly) every opsi-script command.

15.2. Creating opsi-script scripts

You may use every text editor. We recommend to use the jedit editor with integrated opsi-script syntax highlighting.

For testing opsi-script scripts it is a good idea to run them from an interactive started opsi-script. (see: getting-started for more details)

15.2.1. 1. Lection

In the first lection you should just list all files of your c:\windows\system32 directory.

You should use the following opsi-script functions:

- DosInAnIcon

15.2.2. 2. Lection

Extend your script of the first lection by assingning the output of your DosInAnIcon call to a string list

You should use the following opsi-script functions:

- DefStringlist
- getOutStreamFromSection
- setloglevel = 7

15.2.3. 3. Lection

You should determine the number of dll files in your c:\windows\system32 and write this number to the log file.

Extend your script of the second lection by extracting from your file list a new list which contains only the dll files and count them.

You should use the following opsi-script functions:

- getListContaining
- count
- comment
15.2.4. 4. Lection

Is there a `kernel32.dll` at your `c:\windows\system32` and which size has it?

Extend your script of the third lection by extracting from your file list a new string which contains only the directory listing entry of the `kernel32.dll`. Then extract the size entry from this string.

You should use the following opsi-script functions:

- `TakeFirstStringContaining`
- `SplitStringOnWhiteSpace`
- `TakeString`

15.2.5. 5. Lection

Which `kernel32.dll` is bigger the 32Bit or the 64Bit variant?

 Extend your script of the fourth lection by running in different mode for the 32 Bit and 64 Bit part.

You should use the following opsi-script functions:

- `DosInAnIcon winst /64bit`

15.3. Solutions

15.3.1. Solution Lection 1

```opsi
[Actions]
comment "Show all Systemfiles"
DosInAnIcon_Dir

[ DosInAnIcon_Dir]
%systemdrive%
\system%
\dir
```

15.3.2. Solution Lection 2

```opsi
[Actions]
DefStringList $list1$

comment "Show all Systemfiles"
comment "Output from DosInAnIcon is assigned to a list"
set $list1$ = getOutStreamFromSection ("DosInAnIcon_Dir")

[ DosInAnIcon_Dir]
%systemdrive%
\system%
\dir
```

15.3.3. Solution Lection 3
15.3.4. Solution Lection 4

[Actions]
setloglevel = 7
DefVar $DLLCount$
DefStringList $list1$

comment "Show all Systemfiles"
comment "Output from DosInAnIcon is setting to a list"
set $list1$ = getOutStreamFromSection ("DosInAnIcon_Dir")
getListContaining($list1,".dll")

comment "Get a partial list with all strings that match <search string>"
set $list1$ = getlistContaining ($list1",.dll")

comment "Number of DLL-Files"
set $DLLCount$ = count ($list1$)

[DosInAnIcon_Dir]
keysystemdrive
\ cd \keysystem\dir *.

15.3.5. Solution Lection 5

[Actions]
setloglevel = 7
DefVar $dirline$
DefStringList $list1$

comment "Show all Systemfiles"
;DosInAnIcon_Dir
comment "Output from DosInAnIcon is setting to a list"
;DosInAnIcon_Dir
;set $list1$ = getOutStreamFromSection ("DosInAnIcon_Dir")
;set $list64$ = getOutStreamFromSection ("DosInAnIcon_Dir winst /64bit")

comment "Get string kernel32.dll"
set $dirline$ = takeFirstStringContaining ($list1","kernel32.dll")
if $dirline$ = ""
  comment "Kernel32.dll not exist"
else
  set $list10$ = splitStringOnWhiteSpace($dirline$)
  set $dirline$ = takeString (2,$list10$
  comment "Size of Kernel32.dll: "$dirline$ B"
endif

[DosInAnIcon_Dir]
keysystemdrive
\ cd \keysystem\dir *."
[Actions]
setloglevel = 7
DefVar $dirline$
DefString $list32$
DefString $list64$

; search for 32 Bit-Version
comment "Output from DosInAnIcon is setting to a list"
set $list32$ = getOutStreamFromSection("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 $list64$ = getOutStreamFromSection("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 larger than the 32Bit-Version with "$dirline$" Byte"
else
    comment "The 32Bit-Version is "$dirline$" Byte is larger than the 64Bit-Version with "$dirline64$" Byte"
endif

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir *.*
Chapter 16. opsi-script libraries

16.1. opsi-script libraries from uib

Documentation of opsi-script libraries that are part of the opsiscript / opsi-script and are maintained by the uib gmbh

16.1.1. Documentation of opsi library: uib_backend.opsiscript

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Documentation of local function setProductsToUninstall

Definition

setProductsToUninstall($productlist$ : stringlist) : string

Description

Sets for the given list of opsi productIds the action request to uninstall

- Parameter: $productlist$
  - Type: Stringlist - Calltype: CallByValue
  - Parameter $productlist$ Description:
    List of opsi productIds
- 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: 22.08.2019
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
[actions]
DefStringlist $productlist$
set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
if not(stringtobool(setProductsToUninstall($productlist$)))
    comment "call of setProductsToUninstall failed"
endif
```

Documentation of local function setProductsToSetup

Definition

setProductsToSetup($productlist$ : stringlist) : string
Description
Sets for the given list of opsi productIds the action request to setup (try 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) Works without resolving dependencies for use in WAN mode while withdependencies is missing in opsiclientd

Author: detlef oertel
Date: 05.11.2018
Email: d.oertel@uib.de
Version: 1.0
Copyright: AGPLv3

Example:

```plaintext
[actions]
DefStringlist $productlist$
set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
if not(stringtobool(setProductsToSetup($productlist$)))
  comment "call of setProductsToSetup failed"
endif
```

Documentation of local function handle_setup_after_property

Definition
handle_setup_after_property($propname$ : string) : string

Description
The given parameter $propname$ has to be the name of a property that holds a list of productIds The function takes this list If it is only one string and this string contains , it will be splitted to a list Sets for the given list of opsi productIds the action request to setup (try also resolving the dependencies)

- Parameter: $propname$
  - Type: String - Calltype: CallByValue
  - Parameter $propname$ 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: 26.05.2020
Email: d.oertel@uib.de
Version: 1.0
Copyright: AGPLv3
Example:

```plaintext
[actions]
DefVar $propname$
if not(stringtobool(handle_setup_after_property($propname$)))
    comment "call of handle_setup_after_property failed"
endif
```

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:

```plaintext
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
```
Documentation of local function delOpsiPoc

Definition

\[
\text{delOpsiPoc}($\text{donotdelList} : \text{stringlist}) : \text{string}
\]

Description

Delete all productOnClientObjects for this client and only for localboot products and not for products that are included in the $\text{donotdelList}$ parameter

- Parameter: $\text{donotdelList}$
  - 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:

```plaintext
set $\text{opsiMetaDataFile} = $\text{targetDir} + \"\poc.json\"
if Fileexists($\text{opsiMetaDataFile})
    comment "Delete existing meta data ...."
    set $\text{tmplist} = \text{createStringList}(\"\text{opsi-vhd-tester}\",\"\text{opsi-vhd-control}\",\"\text{opsi-vhd-auto-upgrade}\")
    if stringToBool(delOpsiPoc($\text{tmplist}$))
        comment "Delete existing meta data successful finished"
    else
        LogError "Delete existing meta data failed"
        endif
    endif
else
    comment "No meta data existing - creating it...."
    if stringToBool(backupOpsiPoc($\text{opsiMetaDataFile}$))
        comment "Backup meta data successful finished"
    else
        LogError "Backup meta data failed"
        endif
    endif
endif
```

Documentation of local function backupOpsiPoc

Definition

\[
\text{backupOpsiPoc}($\text{filename} : \text{string}) : \text{string}
\]

Description

Get all localboot productOnClient objects for this client and write it to the json file $\text{filename}$

- Parameter: $\text{filename}$
  - Type: String - Calltype: CallByValue
  - Parameter $\text{filename}$ Description: Complete name of the file to create
Documentation of local function **restoreOpsiPoc**

**Definition**

`restoreOpsiPoc($filename$: string) : string`

**Description**

Load productOnClient objects from $filename$ and write it 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:

```bash
see delOpsiPoc
```

---

Documentation of local function **getInstalledLocalbootProductsWithVersion**

**Definition**

`getInstalledLocalbootProductsWithVersion(ref $productlist$: stringlist) : string`

---

Chapter 16. opsi-script libraries
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:

```plaintext
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 $tmplist$ = getKeyList($installedproducts$)
set $possibleproducts$ = getSubListByKey($tmplist$, $possibleproducts$)
for %aktprod% in $installedproducts$ do sub_find_updatable_products
```

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
16.1.2. Documentation of opsi library: uib_bootutils.opsiscript

Documentation of local function delFromWindowsBootmanager

Definition

delFromWindowsBootmanager($bootlabel$ : string) : string

Description

Deletes the boot entry given by the parameter $bootlabel$ from Windows boot manager by using bcdedit

- Parameter: $bootlabel$
  - Type: String - Calltype: CallByValue
  - 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: AGPLv3

Example:

```opsi
[actions]
DefStringlist $productlist$
set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
if not(stringtobool(setProductsToSetup($productlist$)))
  comment "call of setProductsToSetup failed"
endif
```
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] [getUefiBcdbootGuid]

**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")
```

---

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] [getUefiBcdbootGuid]

**Author:** detlef oertel

**Date:** 17.5.2018

**Email:** d.oertel@uib.de
Example:

```bash
if runningonUefi
    set $peuefiguid$ = getUefiBcdbootGuid("opsitempwinpe")
    set $exitcode$ = getlastexitcode
    if $exitcode$ == "0"
        if not ($peuefiguid$ == ")
            shellCall("bcdedit /delete "$peuefiguid$")
            endif
        endif
    endif
endif
```

**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`

**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`
### Documentation of local function `getDiskUuid`

**Definition**

```plaintext
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"

---

**Example:**

```plaintext
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
```
Example:

```powershell
for $disk = "0" to calculate($diskcount1+" " -1) do set $aktdisklist$ = addtolist($aktdisklist$,getDiskUuid("%disk%", "x:"),="%disk%")
set $disk$ = getvalue($diskuuid$, $aktdisklist$)
```

**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 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`)
Example:

```c
Message "Enable PE partition...."  
if not (stringToBool(enablePEPartition($disknumber$, $swapPartitionNumber$, $pepartletter$, $useGpt$)))
    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: 1.0.2
- Copyright: AGPLv3

Documentation of local function isMsiExitcodeFatal

Definition

```c
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 msiexec

- 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**: Windows only
- **References**: [isMsExitcodeFatal_short] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExitcodeFatal] [isNsisExitcodeFatal]
- **Author**: detlef oertel
- **Date**: 19.9.2018
- **Email**: d.oertel@uib.de
- **Version**: 1.0.2
- **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
```

**Documentation of local function** isMsiExitcodeFatal_short

**Definition**

```
isMsiExitcodeFatal_short($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string
```

**Description**

Evaluates the given $exitcode$ as MS Error and and gives back a resulting error message on $ErrorString$ if the exit code is well known. For full list of exit dodes use isMsiExitcodeFatal 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 ms
- **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.
Documentation of local function **isMsExitcodeFatal_short**

**Definition**

```c
isMsExitcodeFatal_short($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string
```

**Description**

- Parameter: `$exitcode$
  - Type: String - Calltype: CallByValue
- Parameter: `$allowRebootRequest$
  - Type: String - Calltype: CallByValue
- Parameter: `$errorstring$
  - Type: String - Calltype: CallByReference

**Author:** detlef oertel
**Date:** 14.05.2019
**Email:** d.oertel@uib.de
**Version:** 1.0.2
**Copyright:** AGPLv3

Documentation of local function **isAdvancedMsiExitcodeFatal**

**Definition**

```c
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 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 isMsExitcodeFatal 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: Windows only

* References: [isMsiExitcodeFatal] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExitcodeFatal] [isNsisExitcodeFatal]

• Author: detlef oertel
• Date: 14.05.2019
• Email: d.oertel@uib.de
• Version: 1.0.2
• Copyright: AGPLv3

Documentation of local function isInnoExitcodeFatal

Definition
isInnoExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string

Description
Evaluates the given Sexticode$ as Inno 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 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.
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 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 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**: Windows only

- **References**: [isMsiExitcodeFatal] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExitcodeFatal] [isNsisExitcodeFatal]
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: Windows only
- References: [isMsiExitcodeFatal] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExitcodeFatal] [isNsisExitcodeFatal]
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0.2
- Copyright: AGPLv3

Documentation of local function isGenericExitcodeFatal

Definition

isGenericExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string

Description

Evaluates the given $exitcode$ as Generic 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

Chapter 16. opsi-script libraries 262 / 281
Parameter `$exitcode$` Description: Exit code given

- **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"
  - **References:** `[isMsiExitcodeFatal]` `[isAdvancedMsiExitcodeFatal]` `[isInnoExitcodeFatal]` `[isInstallshieldExitcodeFatal]` `[isNsisExitcodeFatal]`

- **Author:** detlef oertel
- **Date:** 16.2.2021
- **Email:** d.oertel@uib.de
- **Version:** 1.0.2
- **Copyright:** AGPLv3

---

**Documentation of local function boolToGenericExitcode**

**Definition**

```
boolToGenericExitcode($BoolString$ : string) : string
```

**Description**

Converts the given `$BoolString$` as Generic Exitcode: "True": returns "0" "False": returns "1" is used to decide if an `ExitWindows /Reboot` is called or not

- **Parameter: `$boolstring$`**
  - Type: `String` - Calltype: `CallByValue`
  - **Parameter `$boolstring$` Description:** boolean string given
  - **Parameter `$boolstring$` Advice:** if not "true" default result is "1"

- **Returns:** Returns string "0" or "1"
- **OnError:** Returns string "1"

- **Author:** detlef oertel
- **Date:** 08.3.2021
- **Email:** d.oertel@uib.de
- **Version:** 1.0.2
- **Copyright:** AGPLv3
16.1.4. Documentation of opsi library: uib_strlistutils.opsiscript

• Email: d.oertel@uib.de
• Version: 1.0
• Copyright: AGPLv3

Documentation of local function compareLists

Definition

compareLists($list1$ : stringlist, $list2$ : stringlist) : string

Description
Checks if two string lists are completely identical 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 identical. Returns string "false" if the given lists are not identical
• 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

16.1.5. Documentation of opsi library: uib_driver_install.opsiscript

• Email: d.oertel@uib.de
• Version: 1.0
• Copyright: AGPLv3
Documentation of local function install_driver_recursive_from_dir

Definition

install_driver_recursive_from_dir($driverdir$ : string, ref $errorstrings$ : stringlist) : string

Description

Sets for the given list of opsi productIds the action request to setup (also resolving the dependencies)

- Parameter: $driverdir$
  - Type: String - Calltype: CallByValue
- Parameter: $errorstrings$
  - Type: Stringlist - Calltype: CallByReference
- 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
```

Documentation of local function setFirewallExceptionsTcpUdp4product4profile

Definition

setFirewallExceptionsTcpUdp4product4profile($rulename$ : string, $fwprofile$ : string, $path2product$ : string) : void

Description

- Parameter: $rulename$
  - Type: String - Calltype: CallByValue
- Parameter: $fwprofile$
  - Type: String - Calltype: CallByValue
- Parameter: $path2product$
  - Type: String - Calltype: CallByValue
- Author: detlef oertel
- Date: 27.08.2013
- Email: d.oertel@uib.de
- Version: 1.0
16.1.6. Documentation of opsi library: uib_reboot_lib.opsiscript

Documentation of local function prepare_uefi_boot_to

Definition

prepare_uefi_boot_to($prefer_boot_option$ : string) : void

Description

if this is an uefi OS, then prepare the uefi nextboot to the stored uefi bootlabel that belongs to the boot target $prefer_boot_option$

- Parameter: $prefer_boot_option$
  - Type: String - Calltype: CallByValue
  - Parameter $prefer_boot_option$ Description: network or "local-stored-bootimage" defaults to network if empty

- Returns: Returns string "void"
- OnError: nothing
- Author: detlef oertel
- Date: 20.05.2020

Example:

```powershell
(actions)
prepare_uefi_boot_to('network')
ExitWindows /Reboot
```

16.1.7. Documentation of opsi library: uib_state_and_condition.opsiscript

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Documentation of local function isRebootPending

Definition

isRebootPending() : string

Description

Checks if a Windows or Linux client needs a reboot

- Returns: Returns string "true" if reboot is required
- OnError: Returns string "false"
- Links:
  - https://ilovepowershell.com/2015/09/10/how-to-check-if-a-server-needs-a-reboot/
  - https://serverfault.com/questions/122178/how-can-i-check-from-the-command-line-if-a-reboot-is-required-on-rhel-or-centos
  - https://megamorf.gitlab.io/2019/06/10/check-if-reboot-is-required-after-installing-linux-updates.html
Example:

```python
[actions]
if stringToBool(isRebootPending())
    comment "we need a reboot"
else
    comment "we do not need a reboot"
endif
```

**Documentation of local function** `get_dotnet4_version`

**Definition**

`get_dotnet4_version() : string`

**Description**

Checks which dotnet 4.x is installed

- Returns: Returns string with the version number (aka "4.6") or empty string if no dotnet 4 is installed
- OnError: Returns string ""
- Author: detlef oertel
- Date: 27.08.2019
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```python
[actions]
if ("" = get_dotnet4_version()) or CompareDotSeparatedNumbers(get_dotnet4_version(), ",<","4.6")
    comment "No dotnet 4 >= 4.6 installed"
else
    dotnet 4 >= 4.6 installed"
endif
```

**Documentation of local function** `isdotnet_3_5_installed`

**Definition**

`isdotnet_3_5_installed() : string`

**Description**

Checks if a dotnet 3.5 is installed

- Returns: Returns string "true" dotnet 3.5 is installed
- OnError: Returns string "false"
- Author: detlef oertel
Example:

```actions
if stringToBool(isdotnet_3_5_installed())
    comment "dotnet 3.5 is installed"
else
    comment "No dotnet 3.5 found"
endif
```

Documentation of local function getOpsiNetworkMedium

**Definition**

```getOpsiNetworkMedium() : string```

**Description**

- Author: detlef oertel
- Date: 27.08.2019
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Documentation of local function isVhdMode

**Definition**

```isVhdMode() : string```

**Description**

check if this installation is a vhd-reset installation

- Returns: Returns string "true" if vhd is detected
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)

**Example:**

```actions
if stringToBool(isVhdMode())
    comment "vhd mode detected"
endif
```
Documentation of local function **isOliMode**

**Definition**

isOliMode() : string

**Description**

check if this installation is a *opsi local image* (oli) installation

- Returns: Returns string "true" if oli is detected
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 22.08.2019
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```plaintext
[actions]
if stringtobool(isOliMode())
  comment "oli mode detected"
endif
```

16.1.8. Documentation of opsi library: **uib_lin_install.opsiscript**

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Documentation of local function **cleanupPackageSystem**

**Definition**

cleanupPackageSystem() : void

**Description**

reads repo list und try to repair well known problems should be called after modifying the repo list or after failed installs

- Returns: nothing
- OnError: error counter increased ; Error messages in the log
- Author: detlef oertel
- Date: 19.08.2020
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:
importlib "uib_lin_install"

DefStringlist $packages$
DefVar $installresult$
DefStringlist $errorList$
DefVar $fatal_error$
DefVar $result_string$

comment "update and clean package system"
cleanupPackageSystem()

comment "install pending updates"
set $result_string$ = installupdates()

comment "install new needed packages"
set $packages$ = CreateStringlist("apt-release","cifs-utils","xterm", "dnsutils", "leafo","openssl","pkg-config","desktop-file-utils","libnotify-bin","libzypp-0.0")

comment "if we are on debian / ubuntu we can use debinstall()"
set $installresult$ = debinstall($packages$)
if not(stringtobool($installresult$))
    if waitForPackageLock("300", "false")
        comment "we got the package lock."
    else
        LogError "could not get Package Lock"
    endif
    cleanupPackageSystem()
endif
if not(stringtobool($installresult$))
    LogError "failed dependent packages"
    Message "failed dependent packages"
    isFatalError "failed dependent packages"
    set $fatal_error$ = "true"
    setloglevel = 6
    set $errorList$ = addtolist($errorList$, " failed dependent_packages")
endif
endif

Documentation of local function installupdates

Definition

installupdates() : string

Description

try to install pending updates from the known repositories should be called after modifying the repo list or after failed installs

• Returns: nothing
• OnError: Returns string "false"; error counter increased; Error messages in the log
• Author: detlef oertel
• Date: 19.08.2020
• Email: d.oertel@uib.de
• Version: 1.0
• Copyright: AGPLv3

Example:

see: cleanupPackageSystem()

Documentation of local function runCommandWithList

Definition

runCommandWithList($command$ : string, $list$ : stringlist) : string

Example:

see: cleanupPackageSystem()
Documentation of local function debinstall

Definition

debinstall($packagelist$ : stringlist) : string

Description

try to install the packages given by $packagelist$

• Parameter: $packagelist$
  - Type: Stringlist - Calltype: CallByValue

Example:

see: cleanupPackageSystem()

Documentation of local function redinstall

Definition

redinstall($packagelist$ : stringlist) : string

Description

try to install the packages given by $packagelist$
**Parameter:** $packagelist$

- **Type:** Stringlist - **Calltype:** CallByValue

- **Parameter description:**
  stringlist with packages to install

- **Returns:** Returns string "true" if all is ok

- **OnError:** Returns string "false"; error counter increased; Error messages in the log

- **References:** [cleanupPackageSystem] [installupdates] [debinstall] [suseinstall] [ucsinstall]

- **Author:** detlef oertel

- **Date:** 19.08.2020

- **Email:** d.oertel@uib.de

- **Version:** 1.0

- **Copyright:** AGPLv3

**Example:**

```bash
see: cleanupPackageSystem()
```

---

**Documentation of local function suseinstall**

**Definition**

```bash
suseinstall($packagelist$ : stringlist) : string
```

**Description**

try to install the packages given by $packagelist$

- **Parameter:** $packagelist$
  
  - **Type:** Stringlist - **Calltype:** CallByValue

  - **Parameter description:**
    stringlist with packages to install

- **Returns:** Returns string "true" if all is ok

- **OnError:** Returns string "false"; error counter increased; Error messages in the log

- **References:** [cleanupPackageSystem] [installupdates] [debinstall] [redinstall] [ucsinstall]

- **Author:** detlef oertel

- **Date:** 19.08.2020

- **Email:** d.oertel@uib.de

- **Version:** 1.0

- **Copyright:** AGPLv3

**Example:**

```bash
see: cleanupPackageSystem()
```
Documentation of local function **ucsinstall**

**Definition**

```
ucsinstall($packagelist$ : stringlist) : string
```

**Description**

try to install the packages given by $packagelist$

- **Parameter: $packagelist$**
  - Type: Stringlist - Calltype: CallByValue
  - Parameter $packagelist$ Description: stringlist with packages to install

- **Returns:** Returns string "true" if all is ok

- **OnError:** Returns string "false"; error counter increased ; Error messages in the log

- **References:** [cleanupPackageSystem] [installupdates] [debinstall] [redininstall] [suseinstall]

- **Author:** detlef oertel
- **Date:** 19.08.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

**Example:**

```
see: cleanupPackageSystem()
```

---

Documentation of local function **genericLinInstall**

**Definition**

```
genericLinInstall($packagelist$ : stringlist) : string
```

**Description**

try to determine the Linux familily and try to install the packages given by $packagelist$

- **Parameter: $packagelist$**
  - Type: Stringlist - Calltype: CallByValue
  - Parameter $packagelist$ Description: stringlist with packages to install

- **Returns:** Returns string "true" if all is ok

- **OnError:** Returns string "false"; error counter increased ; Error messages in the log

- **References:** [cleanupPackageSystem] [installupdates] [debinstall] [redininstall] [suseinstall]

- **Author:** detlef oertel
- **Date:** 08.03.2021
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

**Example:**
Documentation of local function **linuxInstallOneOf**

**Definition**

`linuxInstallOneOf($packagelist$ : stringlist) : string`

**Description**

try to install any package given by $packagelist$ This can be used specifying a package with different names for different linux distributions.

- **Parameter:** $packagelist$
  - Type: Stringlist - Calltype: CallByValue
  - Parameter $packagelist$ Description:
    stringlist with packages to install
- **Returns:** Returns string *true* if one package was successfully installed
- **OnError:** Returns string *false*
- **References:** [isOneInstalled] [getLinuxCommand]
- **Author:** nils doerrer
- **Date:** 16.11.2020
- **Email:** d.oertel@uib.de
- **Version:** 1.0
- **Copyright:** AGPLv3

**Example:**

```plaintext
importlib "uib_lin_install"
if isOneInstalled(createStringList("lsusb", "usbutils")) = "False"
    message "installing lsusb or usbutils"
    set $success$ = linuxInstallOneOf(createStringList("lsusb", "usbutils"))
endif
```

Documentation of local function **isOneInstalled**

**Definition**

`isOneInstalled($packagelist$ : stringlist) : string`

**Description**

check for installation status and return if any of $packagelist$ exists This can be used to check a package with different names for different linux distributions.

- **Parameter:** $packagelist$
  - Type: Stringlist - Calltype: CallByValue
  - Parameter $packagelist$ Description:
    stringlist with packages to check
- **Returns:** Returns string *true* if one specified package is installed
- **OnError:** Returns string *false*
Documentation of local function getLinuxCommand

Definition

getLinuxCommand($type$ : string) : string

Description

Determine package manager and return command.

- Parameter: $type$
  - Type: String - Calltype: CallByValue
  - Parameter $type$ Description:
    type of desired command install, check, localpackage

- Returns: Package manager command according to type
- OnError: Returns string False

References: [linuxInstallOneOf] [isOneInstalled]

Author: nils doerrer, d.oertel
Date: 14.01.2021
Email: d.oertel@uib.de
Version: 1.0
Copyright: AGPLv3

Documentation of local function getLinuxCommandAgnostic

Definition

getLinuxCommandAgnostic($type$ : string) : string

Description

Determine package manager and return command.

- Parameter: $type$
  - Type: String - Calltype: CallByValue
  - Parameter $type$ Description:
    type of desired command install or check

- Returns: Package manager command according to type
- OnError: Returns string False

References: [linuxInstallOneOf] [isOneInstalled] [getLinuxCommand]

Author: nils doerrer
Documentation of local function linuxInstallOneFile

Definition

linuxInstallOneFile($packagefile$ : string) : string

Description

try to install the local file package given by $packagefile$. This can be used specifying a package with different names for different Linux distributions.

- Parameter: $packagefile$
  - Type: String - Calltype: CallByValue
- Returns: Returns string 0 if one package was successfully installed
- OnError: Returns string -1
- References: [isOneInstalled] [getLinuxCommand]

Example:

```plaintext
importlib "uib_lin_install"
if stringToBool(linuxInstallOneFile("/tmp/dummy.deb")
  comment "success"
endif
```

Documentation of local function linuxRemoveOnePackage

Definition

linuxRemoveOnePackage($packagename$ : string) : string

Description

try to remove the package given by $packagename$. This can be used specifying a package with different names for different Linux distributions.

- Parameter: $packagename$
  - Type: String - Calltype: CallByValue
  - Parameter $packagename$ Description: string with the name of a package to remove
- Returns: Returns string 0 if package was successfully removed or was not installed
Example:

```bash
[Actions]
importlib "uib_lin_install"
if stringToBool(linuxRemoveOnePackage("dummy"))
    comment "success"
endif
```

**Documentation of local function linuxRemoveOneOf**

**Definition**

`linuxRemoveOneOf($packagelist$ : stringlist) : string`

**Description**

try to remove any package given by $packagelist$ This can be used specifying a package with different names for different linux distributions.

- **Parameter:** `$packagelist$`  
  - Type: `Stringlist` - Calltype: `CallByValue`  
    - Parameter `$packagelist$` Description: stringlist with packages to install

- **Returns:** Returns string `True` if one package was successfully installed
- **OnError:** Returns string `False`

**References:** [isOneInstalled] [getLinuxCommand]

**Author:** nils doerrer / detlef oertel

**Date:** 16.11.2020

**Email:** d.oertel@uib.de

**Version:** 1.0

**Copyright:** AGPLv3

Example:

```bash
[Actions]
importlib "uib_lin_install"
if isOneInstalled(createStringList("lsusb", "usbutils")) = "True"
    message "installing lsusb or usbutils"
    set $success$ = linuxRemoveOneOf(createStringList("lsusb", "usbutils"))
endif
```

**16.1.9. Documentation of opsi library: uib_macosinstalllib.opsiscript**

- **Email:** d.oertel@uib.de
Documentation of local function `install_macos_app`

**Definition**

```
install_macos_app($myapp$ : string) : string
```

**Description**

try to install the app given by `$myapp$`

- **Parameter:** `$myapp$
  - Type: String - Calltype: CallByValue
  - Parameter `$myapp$` Description:
    string with path to app to install

- **Returns:** Returns string "true" if all is ok
- **OnError:** Returns string "false"; error counter increased; Error messages in the log
- **References:** [install_macos_pkg] [install_macos_dmg] [install_macos_zip] [install_macos_generic]

**Author:** detlef oertel
**Date:** 28.12.2020
**Email:** d.oertel@uib.de

Example:

```
[Actions]
importlib "uib_macosinstalllib"
DefVar $installfile$
DefVar $installresult$

comment "we have a *.app (directory) and install with install_macos_app"
set $installfile$ = "$scriptpath%/files/my_macos_app.app"
set $installresult$ = install_macos_app($installfile$)
if stringtobool($installresult$)
    comment "installation succeeded"
else
    LogError "installation failed"
endif
```

Documentation of local function `install_macos_pkg`

**Definition**

```
install_macos_pkg($mypkg$ : string) : string
```

**Description**

try to install the pkg file given by `$mypkg$`

- **Parameter:** `$mypkg$
  - Type: String - Calltype: CallByValue
  - Parameter `$mypkg$` Description:
string with path to pkg file to install

- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"; error counter increased; Error messages in the log
- References: [install_macos_app] [install_macos_dmg] [install_macos_zip] [install_macos_generic]
- Author: detlef oertel
- Date: 28.12.2020
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```python
importlib "uib_macosinstalllib"
DefVar $installfile$
DefVar $installresult$

comment "we have a *.pkg and install with install_macos_pkg"
set $installfile$ = "%scriptpath%/files/my_macos_app.pkg"
set $installresult$ = install_macos_pkg($installfile$)
if stringtobool($installresult$)
    comment "installation succseeded"
else
    LogError "installation failed"
endif
```

Documentation of local function install_macos_dmg

Definition

```
install_macos_dmg($mydmg$ : string) : string
```

Description

try to install the dmg file given by $mydmg$

- Parameter: $mydmg$
  - Type: String - Calltype: CallByValue
  - Parameter $mydmg$ Description:
    string with path to dmg file to install
  - Returns: Returns string "true" if all is ok
  - OnError: Returns string "false"; error counter increased; Error messages in the log
  - References: [install_macos_app] [install_macos_pkg] [install_macos_zip] [install_macos_generic]
  - Author: detlef oertel
  - Date: 28.12.2020
  - Email: d.oertel@uib.de
  - Version: 1.0
  - Copyright: AGPLv3
importlib "uib_macosinstalllib"

DefVar $installfile$
DefVar $installresult$

comment "we have a *.dmg and install with install_macos_dmg"
set $installfile$ = "$scriptpath%/files/my_macos_app.dmg"
set $installresult$ = install_macos_dmg($installfile$)
if stringtobool($installresult$)
    comment "installation succseeded"
else
    LogError "installation failed"
endif

Documentation of local function install_macos_zip

Definition

install_macos_zip($myzip$ : string) : string

Description

try to install the zip file given by $myzip$ unzips the file and try to find a installable part (*.app, *.pkg, *.dmg) and try to install this

- Parameter: $myzip$
  - Type: String - Calltype: CallByValue
  - Parameter $myzip$ Description:
    string with path to zip file to install

- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"; error counter increased ; Error messages in the log
- References: [install_macos_app] [install_macos_dmg] [install_macos_pkg] [install_macos_generic]
- Author: detlef oertel
- Date: 28.12.2020
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

importlib "uib_macosinstalllib"

DefVar $installfile$
DefVar $installresult$

comment "we have a *.zip and install with install_macos_zip"
set $installfile$ = "$scriptpath%/files/my_macos_app.zip"
set $installresult$ = install_macos_zip($installfile$)
if stringtobool($installresult$)
    comment "installation succseeded"
else
    LogError "installation failed"
endif

Documentation of local function install_macos_generic

Definition

install_macos_generic($myfile$ : string) : string
Description

try to install the file given by $myfile$. Checks if the file is a well known installable (*.app, *.pkg, *.dmg, *.zip) and try to install this

• Parameter: $myfile$
  - Type: String - Calltype: CallByValue
  - Parameter $myfile$ Description: string with path to pkg file to install

• Returns: Returns string "true" if all is ok

• OnError: Returns string "false"; error counter increased ; Error messages in the log

• References: [install_macos_app] [install_macos_dmg] [install_macos_zip] [install_macos_pkg]

• Author: detlef oertel
• Date: 28.12.2020
• Email: d.oertel@uib.de
• Version: 1.0
• Copyright: AGPLv3

Example:

```
see: install_macos_generic

importlib "uib_macosinstalllib"

DefVar $installfile$
DefVar $installresult$

comment "we have a *.* and install with install_macos_generic"
set $installfile$ = "%scriptpath%/files/opsi-script.app"
set $installresult$ = install_macos_generic($installfile$)
if stringtobool($installresult$)
    comment "installation succseeded"
else
    Error "installation failed"
endif
```