opsi-winst / opsi-script Manual (4.12.0)

uib gmbh
Bonifaziusplatz 1b
55118 Mainz
Tel.:+49 6131 275610
www.uib.de
info@uib.de

Stand: December 17, 2018

Contents

Chapter 1

Copyright

Chapter 2

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

2.1 Global text constants

2.1.1 System directories

2.1.1.1 System directories [W]:

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

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

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

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

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

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

`%Systemdrive%` : *c:*

`%ProfileDir%` :
NT5: *c:\Documents and Settings*
NT6: *C:\users\*

see also: Section 8.2.3.1

2.1.2 Common (AllUsers) directories [W]:

`%AllUsersProfileDir%` or `%CommonProfileDir%` :
NT5: *c:\Documents and Settings\All Users*
NT6: *C:\Users\Public*

`%CommonStartMenuPath%` or `%CommonStartmenuDir%` :
NT5: *c:\Documents and Settings\All Users\Startmenu*
NT6: *C:\ProgramData\Microsoft\Windows\Start Menu*

`%CommonAppdataDir%` :
NT5: *c:\Documents and Settings\All Users\Application Data*
NT6: *C:\ProgramData*

`%CommonDesktopDir%`
NT5: *c:\Documents and Settings\All Users\Desktop*
NT6: *C:\Users\Public\Desktop*

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

`%CommonProgramsDir%`

see also : Section 8.2.3.2

2.1.3 Default User directories [W]:

`%DefaultUserProfileDir%` //since 4.11.1.1

see also : Section 8.2.3.3

2.1.4 Current user directories [W]:

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

`%AppdataDir%` or `%CurrentAppdataDir%` : //since 4.10.8.13
NT5: *c:\Documents and Settings\%USERNAME%\Application Data* NT6: *c:\users\%USERNAME%\Appdata\Roaming*

`%CurrentStartmenuDir%` //since 4.10.8.13

`%CurrentDesktopDir%` //since 4.10.8.13

`%CurrentStartupDir%` //since 4.10.8.13

`%CurrentProgramsDir%` //since 4.10.8.13

`%CurrentSendToDir%` //since 4.10.8.13

`%CurrentProfileDir%` //since 4.11.2.1

see also : Section 8.2.3.4

2.1.5 /AllNtUserProfiles directory constants [W]:

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

see also : Section 8.2.3.5

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

`%ScriptPath% or %ScriptDir%`

`%ScriptDrive%`

`%WinstDir%`

`%WinstVersion%` //4.10.8.3

`%Logfile%`

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

`%opsiTmpDir%` : `c:\opsi.org\tmp` // since 4.11.4.3

`%opsiLogDir%` : `c:\opsi.org\log` // since 4.11.4.3

`%opsidata%` : `c:\opsi.org\data` // since 4.12.0.12

`%opsiapplog%` : `c:\opsi.org\applog` // since 4.12.0.12

see also : Section 8.2.4

2.1.7 Network informations [W/L]:

`%Host%` : value of environment variable HOST.

`%PCName%`: value of environment variable PCNAME, or if absent of COMPUTERNAME.

`%Username%` : Name of actual user.

`%IPName%` : The dns name of the pc. Usually identical with the netbios name and therefore with `%PCName%` besides that the netbios names uses to be uppercase.

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

see also : Section 8.2.5

2.1.8 Service Data [W/L]

`%HostID%` : FQDN of the client

`%opsiserviceURL%`

`%opsiServer%`

`%opsiDepotId%` //since 4.11.4

`%opsiserviceUser%` FQDN used for the connection to the opsi-config-server

`%opsiservicePassword%`

`%installingProdName%`: productid //since 4.10.8

`%installingProdVersion%`: product version //since 4.10.8

`%installingProduct%` : productid (deprecated)

see also : Section 8.2.6

2.2 In Primary Sections

2.2.1 Kinds of Primary Sections [W/L]:

`[Initial]`

`[Actions]`

`[sub`<identifier>`]`

`sub` <file name>

`[ProfileActions]` [W]

see also : Chapter 9

2.2.2 Winst control [W/L]:

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

`LogLevel` (deprecated) see also : Section 9.2.1

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

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

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

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

see also : [opsi-script-configs_ScriptErrorMessages]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.3 Variables [W/L]:

2.2.3.1 Strings

`DefVar` <variable name>

`Set` <variable name> `=` <value>

see also : Section 8.3

2.2.3.2 Stringlists

`DefstringList` <variable name>

see also : Section 8.4

2.2.4 Functions

2.2.4.1 String functions

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

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

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

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

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

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

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

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

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

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

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

see also : [GetValueFromInifile]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

see also : [DemandLicenseKey]

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

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

see also : [FreeLicense]

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

`GetLoggedInUser` //since 4.11.1.2 see also : [GetLoggedInUser]

`GetUsercontext` //since 4.11.1.2 see also : [getLastExitCode]

`GetScriptMode` possible values *Machine*,*Login* //since 4.11.2.1 see also : [GetUsercontext]

`saveVersionToProfile` - save productversion-packageversion to local profile //since 4.11.2.1 see also : [saveVersionToProfile]

`readVersionFromProfile` : string - read productversion-packageversion from local profile //since 4.11.2.1 see also : [readVersionFromProfile]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`GetNtVersion` Deprecated - please use `GetMsVersionInfo` [W] see also : [GetMsVersionInfo]

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

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

2.2.4.2 String list functions

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

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

see also : [splitString]

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

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

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

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

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

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

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

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

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

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

see also : [getOutStreamFromSection]

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

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

see also : [shellCall_list]

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

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

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

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

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

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

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

see also : [forInDo]

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

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

`getRegistryKeyList32(`<regkey>`)` //since 4.11.3 [W] see also : [getRegistryKeyList32] `getRegistryKeyList64(<regkey>)` //since 4.11.3 [W] see also : [getRegistryKeyList64]

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

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

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

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

`getProfilesDirList` //since 4.11.3.2 [W/L] see also : [getProfilesDirList]

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

`GetMSVersionMap` [W] see also : [GetMSVersionMap]

`getLinuxVersionMap` //since 4.11.4 [L]
keys are (example):

```
Distributor ID=Ubuntu
Description=Ubuntu 12.04.2 LTS
Release=12.04
Codename=precise
kernel name=Linux
node name=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( <file name> )` [W] see also : [getFileInfoMap]

`getProductMap` // since 4.11.2.4 [W/L]
keys are: id, name, description, advice, productversion, packageversion, priority, installationstate, lastactionrequest, lastactionresult, installedversion, installedpackage, installedmodificationtime,actionrequest
see also : [getProductMap]

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

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

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

`getHWBiosInfoMap` //since 4.11.4 [W/L] see also : [getHWBiosInfoMap]

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

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

see also : [createStringList]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.4.3 Boolean operators and functions

see also : Section 9.14.2

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Example:

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

`opsiLicenseManagementEnabled` [W/L] see also : [opsiLicenseManagementEnabled]

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.4.4 Misc functions

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

`requiredWinstVersion` <relation operator> <version> [W/L] see also : [requiredWinstVersion]

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

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

2.2.4.5 Flow control

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

Syntax:

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

Example:

```
Set $NTVer$ = GetMsVersionInfo
if ( $NTVer$ >= "6" )
     sub_install_win7
else
  if ( $NTVer$ = "5.1" )
```

```
    sub_install_winXP
  else
    stop "not a supported OS-Version"
  endif
endif
```

for - to - do Statement //since 4.11.5 [W/L] see also : [ForToDo]

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

Example:

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

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

Syntax:

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

Example:

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

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

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

`isSuccess` //since 4.11.3.7 [W/L] see also : [isSuccess]

`isSuspended` //since 4.11.4.1 [W/L] see also : [isSuspended]

`noUpdateScript` //since 4.11.3.7 [W/L] see also : [noUpdateScript]

`ExitWindows /Reboot` [W/L] see also : [Reboot]

`ExitWindows /ImmediateReboot` [W/L] see also : [ImmediateReboot]

`ExitWindows /ImmediateLogout` [W] see also : [ImmediateLogout]

`ExitWindows /ShutdownWanted` [W] see also : [ShutdownWanted]

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

`sleepSeconds` <Integer> or <string> : noresult [W/L] see also : [sleepSeconds]

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

2.3 Secondary Sections

2.3.1 Winbatch [W/L]

see also : Section 10.9

Function: execute programs via operating system API

`[WinBatch`<identifier>]

Modifier:

`/LetThemGo`

`/WaitForProcessEnding` "<program.exe>"

`/TimeOutSeconds` <seconds>

`/WaitForWindowAppearing` <window title> (*does not work with 64 Bit programs*) [W]

`/WaitForWindowVanish` <window title> (*does not work with 64 Bit programs*) [W]

`/RunElevated` // since 4.11.3: only at >= NT6 ; no network access [W]

`/RunAsLoggedOnUser` // since 4.11.3.5 ; works only inside *userLoginScripts* [W]

`/32Bit` //since 4.11.3.5 [W]

`/64Bit` //since 4.11.3.5 [W]

`/SysNative` //since 4.11.3.5 [W]

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

see also : Section 10.10

Function: Execute section via cmd.exe

`[DosBatch`<identifier>`]` <optional parameters> <winst <modifier>>

`[DosInAnIcon`<identifier>`]` <optional parameters> <winst <modifier>>

`[ShellBatch`<identifier>`]` <optional parameters> <winst <modifier>>

`[ShellInAnIcon`<identifier>`]` <optional parameters> <winst <modifier>>

Modifier: //since 4.11.1.1

`/32Bit` [W]

`/64Bit` [W]

`/SysNative` [W]

`/showoutput` [W/L] // since 4.11.4.7

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

```
DosInAnIcon_do_64bit_stuff winst /64Bit
```

Commands: see manual

2.3.3 ExecWith [W/L]

see also : Section 10.14

Function: Execute section via any interpreter

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

Modifier:

`/LetThemGo`

`/EscapeStrings`

`/32Bit` //since 4.11.3.5 [W]

`/64Bit` //since 4.11.3.5 [W]

`/SysNative` //since 4.11.3.5 [W]

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

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

Commands: see manual

2.3.4 Files [W/L]

see also : Section 10.1

Function: File Operations

`[Files`<identifier>`]`

Modifier [W]:

`/AllNTUserProfiles`

`/AllNTUserSendTo` [W]

`/32Bit` //since 4.10.8 [W]

`/64Bit` //since 4.10.8 [W]

`/SysNative` //since 4.10.8 [W]

Commands:

`checkTargetPath =` <destination directory> [W/L]

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

some options:

`-s` recursive [W/L]

`-V` version control against targetdir [W]

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

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

`-d` date check [W]

`-u` update [W]

`-x` extract [W]

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

`-n` no overwrite [W]

`-r` copy read only attribute [W]

`-h` follow symlinks [L] //since 4.11.6.14

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

some options: `-s` rekursiv `-f` force

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

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

Works like `delete` but on
`del -s -f c:\not-exists`
if `c:\not-exists` not exists it do not search complete `c:\` for `not-exits`

Example (**you may forget the trailing Backslash**):
`del -sf c:\delete_this_dir`

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

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

`symlink` <existing file> <new file> // since 4.11.5 [W/L]
At Windows `symlink` is only available at NT6 and up.

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

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

2.3.5 Registry [W]

see also : Section 10.11

Function: edit Registry

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

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

Alternative method call (deprecated):
`Registry loadUnicodeTextFile(`<.addreg file>`) /addreg`
import the specified <.addreg file>.

Modifier:

`/AllNTUserDats`

`/32Bit` //since 4.10.8

`/64Bit` //since 4.10.8

`/SysNative` //since 4.10.8

Commands:

`OpenKey` <Key>

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

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

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

Examples for registry types:

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

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

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

`GetMultiSZFromFile` <varname> <file name>

`SaveValueToFile` <varname> <file name>

`DeleteVar` <varname>

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

2.3.6 Patches [W/L]

see also : Section 10.2

Function: edit Ini-files

`[Patches`<identifier>`]` <file name>

Modifier:

`/AllNTUserProfiles` //since 4.11.3 [W]

Commands:

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

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

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

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

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

`del [`<section name>`]` <variable1>

`delsec [`<section name>`]`

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

2.3.7 PatchTextFile [W/L]

see also : Section 10.5

Function: edit text files

`[PatchTextFile`<identifier>`]` <file name>

Modifier:

`/AllNTUserProfiles` //since 4.11.3.4 [W]

Commands:

`Set_Mozilla_Pref` ("<preference type>", "<preference key>", "<preference value>")
preference types are usally:
`pref`, `user_pref`, `lock_pref`

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

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

`AddstringListElement_To_Netscape_User_Pref` (*deprecated*)

`FindLine` <search string>

`FindLine_StartingWith` <search string>

`FindLine_Containing` <search string>

`GoToTop`

`AdvanceLine` [<number of lines>]

`GoToBottom`

`DeleteTheLine`

`AddLine_` <line> or `Add_Line_` <line>

`InsertLine` <line> or `Insert_Line_` <line>

`AppendLine` <line> or `Append_Line` <line>

`Append_File` <file name>

`Subtract_File` <file name>

`SaveToFile` <file name>

`Sorted`

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

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

2.3.8 LinkFolder [W/L]

see also : Section 10.6

Function: Startmenue + Desktop Icons

`[LinkFolder`<identifier>`]`

Commands:

`set_basefolder` <system folder>

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

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

`delete_element` <link name>

`delete_subfolder` <folder path> [W]

The predefined virtual system folders which can be used are at Windows:
`desktop, sendto, startmenu, startup, programs, desktopdirectory,`

`common_startmenu, common_programs, common_startup, common_desktopdirectory`
and at Linux:
`common_programs`,`common_startup`,`desktop`, `startup`

Predefined `link_categories` for Linux:
`AudioVideo`, `Audio`, `Video`, `Development`, `Education`, `Game`, `Graphics`, `Network`, `Office`, `Settings`, `System`, `Utility`

Examples

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

```
[LinkFolder_configed_lin]
set_basefolder common_programs
set_subfolder ""

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

The predefined virtual system folders:
`desktop, sendto, startmenu, startup, programs, desktopdirectory`
are pointing to the folders of the user that the script is running. If you use it in a userLoginScript with the opsi *User Profile Management* extension these virtual folders point to the folder of the user that just had logged in.

shortcut defaults to empty. // since 4.11.6.7
`shortcut` may be a combination of [*shift*,*alt*,*ctrl*] (not case sensitiv) divided by ' , '-,+ an a *Key* or a *Virtual Key Code*.
The *Key* is a letter (*A* - *Z*) or a numeral (*0* - *9*). All other Keys must be given by there *Virtual Key Code* identifier. To get these identifier (as well as the allowed combinations) just use the following helper program:
http://download.uib.de/opsi4.0/helper/showkeys.exe

2.3.9 OpsiServiceCall [W/L]

see also : Section 10.12

Function: opsi-Service access

`[OpsiServiceCall`<identifier>]

Commands: see manual

2.3.10 PatchHosts [W/L]

see also : Section 10.3

Function: hosts-files bearbeiten

`[PatchHosts`<identifier>]

Commands:

`setaddr` <hostname> <IPaddress>

`setname` <IPaddress> <hostname>

`setalias` <hostname> <alias>

`setalias` <IPadresse> <alias>

`delalias` <hostname> <alias>

`delalias` <IPaddress> <alias>

`delhost` <hostname>

`delhost` <ipadresse>

`setComment` <ident> <comment>

2.3.11 XMLPatch [W]

see also : Section 10.7

Function: edit XML files

`[XMLPatch`<identifier>`]`

Commands: see manual

2.3.12 ExecPython [W/L]

see also : Section 10.13

Function: Execute section via python interpreter

`[ExecPython`<identifier>`]`

Commands: see manual

2.3.13 LdapSearch [W]

see also : Section 10.15

Function: read from LDAP

`[LdapSearch`<identifier>`]`

Commands: see manual

2.4 By Topic

2.4.1 Compare related functions [W/L]

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

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

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

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

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

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

2.4.2 Crypt / Hash related functions [W/L]

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

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

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

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

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

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

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

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

2.4.3 Defined Functions and Libraries [W/L]

since 4.12.0.0

Definition

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

Where:

- `DefFunc` is the keyword used to start defining a local function..
- *<func name>* is the freely 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 9.18

`importLib` <string expr> ; import library // since 4.12.0.0
<string expr> : <file name>[.<file extension>][`::`<function name>]
If no *.<file extension>* is given `.opsiscript` is used as default.
If no *::<function name>* is given, all function from the given file will be imported.

<file name> is:

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

- A existing file in `%ScriptPath%/../lib` [W/L]
- A existing file in `%WinstDir%\lib` [W] or `/usr/share/opsi-script/lib` [L]

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

see also : Section 9.19

2.4.4 Encoding related functions [W/L]

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

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

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

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

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

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

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

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

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

see also : Section 6.3

2.4.5 Error / Warning related functions [W/L]

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

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

see also : [ScriptErrorMessages]

see also : [opsi-script-configs_ScriptErrorMessages]

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

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

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

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

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

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

`markErrorNumber` see also : [markErrorNumber]

`errorsOccurredSinceMark` <relation> <integer> : boolean see also : [errorsOccurredSinceMark]

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

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

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

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

`getLastServiceErrorClass` : string see also : [getLastServiceErrorClass]

`getLastServiceErrorMessage` : string see also : [getLastServiceErrorMessage]

2.4.6 File related functions [W/L]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

see also: Section 2.3.4

see also: Section 2.3.7

2.4.7 Ini file related functions [W/L]

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

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

see also : [GetValueFromInifile]

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

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

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

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

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

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

see also: Section 2.3.6

2.4.8 Interaction [W/L]

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

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

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

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

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

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

2.4.9 License Management related functions [W/L]

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

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

see also : [DemandLicenseKey]

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

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

see also : [FreeLicense]

`getLastServiceErrorClass` : string see also : [getLastServiceErrorClass]

`getLastServiceErrorMessage` : string see also : [getLastServiceErrorMessage]

- `opsiLicenseManagementEnabled` : bool see also : [opsiLicenseManagementEnabled]

2.4.10 Logging related functions [W/L]

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

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

see also : [SetLogLevel] see also : [opsi-script-configs_default_loglevel] see also : [opsi-script-configs_force_min_loglevel]

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

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

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

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

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

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

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

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

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

opsi-configs

see also : Section 5.2

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

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

`opsi-script.global.default_loglevel` : intstr ; set the default log level; default: *6* see also : [opsi-script-configs_default_loglevel]

`opsi-script.global.force_min_loglevel` : intstr ; set the minimal loglevel; default: *0* see also : [opsi-script-configs_force_min_loglevel]

`opsi-script.global.ScriptErrorMessages` : boolean ; overwrites the opsi-script internal default; default: false see also : [opsi-script-configs_ScriptErrorMessages]

`opsi-script.global.AutoActivityDisplay` : boolean ; overwrites the opsi-script internal default; default: true see also : [opsi-script-configs_AutoActivityDisplay]

2.4.11 Network related functions [W/L]

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

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

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

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

2.4.12 Number related functions [W/L]

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

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

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

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

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

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

2.4.13 Operating System related functions [W/L]

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

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

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

`GetMSVersionMap : stringlist` [W] see also : [GetMSVersionMap]

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

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

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

`getProfilesDirList : stringlist` //since 4.11.3.2 [W/L] see also : [getProfilesDirList]

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

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

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

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

2.4.14 opsiservicecall and json Related functions [W/L]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

see also: OpsiServiceCall Section 2.3.9

see also : Section 9.6

see also : Section 10.12

2.4.15 opsi related functions [W/L]

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

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

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

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

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

2.4.16 Process and Script Related functions [W/L]

`Killtask` <process name> ` : noresult` [W/L] see also : [Killtask]

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

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

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

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

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

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

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

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

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

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

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

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

see also : [getOutStreamFromSection] see also : Section 10.10.2

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

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

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

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

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

see also: ExecWith sections Section 2.3.3

see also: ShellBatch sections Section 2.3.2

see also: Winbatch sections Section 2.3.1

2.4.17 Registry related functions [W]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

see also: Section 2.3.5

see also : Section 10.11

see also : Chapter 11

2.4.18 String handling functions [W/L]

see also : Section 9.4

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

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

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

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

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

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

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

`getIndexFromListByContaining(`<list> : stringlist,<search string> : string`)` : <number> : string //since 4.12.0.13 [W/L] see also : [getIndexFromListByContaining] `contains(<str>, <substr>) : bool` //since 4.11.3: true if <substr> in <str> [W/L] see also : [contains]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.4.19 Stringlist handling functions [W/L]

see also : Section 9.5

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

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

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

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

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

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

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

see also : Section 9.5.10

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

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

see also : [createStringList]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

see also : [splitString]

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

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

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

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

2.4.20 Time / Date related functions [W/L]

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

`markTime` : noresult
sets a time stamp for the current system time and logs it. see also : [markTime]

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

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

2.4.21 Usercontext / loginscripts related functions [W]:

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

`GetLoggedInUser : string` //since 4.11.1.2 see also : [GetLoggedInUser]

`GetUsercontext : string` //since 4.11.1.2 see also : [GetUsercontext]

`GetScriptMode : string` possible values *Machine*,*Login* //since 4.11.2.1 see also : [GetScriptMode]

`saveVersionToProfile : noresult` - save productversion-packageversion to local profile //since 4.11.2.1 see also : [saveVersionToProfile]

`readVersionFromProfile : string` - read productversion-packageversion from local profile //since 4.11.2.1 see also : [readVersionFromProfile]

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

Chapter 3

Introduction

The open source program *opsi-winst/opsi-script* 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-winst/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-winst/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-winst/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

4.1 Introduction

As of version 4.11.4 there is a Linux port of *opsi-winst* with the name *opsi-script*.

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

In the following section the availability is marked as:

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

4.2 Important differences and hints

opsi-script is a GUI application which needs a running and accessible X Windows.

opsi-script-nogui is a command line version which can run without any GUI.

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

4.3 opsi-script path

According to the Linux Filesystem Hierachy Standard the files that belong to *opsi-script* are not at one place but distributed around the system. So here an overview where to find which part:

- 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`

- config files:
  `/etc/opsi-script`
- varibale files:
  `/var/lib/opsi-client-agent/opsi-script`

4.4 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.

4.5 Linux specific functions

For Linux support there are the following special functions:

- `GetOS` // *Linux* or *Windows_NT* [W/L]

- `getLinuxDistroType` // *debian* or *redhat* or *suse* [L]
- `getLinuxVersionMap` [L]
- `chmod` in Files sections [L]

4.6 Example scripts for Linux

4.6.1 Run on Linux only

```
[Actions]
DefVar $OS$

set $OS$ = GetOS

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

4.6.2 Which Linux Version

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


DefStringList $linuxInfo$

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

Table 4.1: getLinuxVersionMap Result Examples

Distro	**Distributor ID**	**Release**	**Codename**	**Description**
Ubuntu Lucid	Ubuntu	10.04	lucid	
Ubuntu Precise	Ubuntu	12.04	precise	Ubuntu 12.04.5 LTS
Ubuntu Trusty	Ubuntu	14.04	trusty	
Debian 6	Debian	6.0.10	squeeze	Debian GNU/Linux 6.0.10 (squeeze)
Debian 7	Debian	7.6	wheezy	Debian GNU/Linux 7.6 (wheezy)
openSUSE 12.3	openSUSE project	12.3	Dartmouth	openSUSE 12.3 (x86_64)
openSUSE 13.1	openSUSE project	13.1	Bottle	openSUSE 13.1 (Bottle) (x86_64)
SLES11SP3	SUSE LINUX	11	n/a	SUSE Linux Enterprise Server 11 (x86_64)
Fedora20	Fedora	20		
CentOS 6.5	CentOS	6.5	Final	
CentOS 7.0	CentOS	7.0.1406	Core	CentOS Linux release 7.0.1406 (Core)
RedHat 6.5	RedHatEnterpriseServer	6.5	Santiago	Red Hat Enterprise Linux Server release 6.5 (Santiago)
RedHat 7.0	RedHatEnterpriseServer	7.0	Maipo	Red Hat Enterprise Linux Server release 7.0 (Maipo)
UCS 3.2	Univention	3.2-3 errata221	Borgfeld	Univention Corporate Server 3.2-3 errata221 (Borgfeld)

4.6.3 ShellInAnIcon call

```
[Actions]

ShellInAnIcon_ls

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

It'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.

4.6.4 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_repo_deb "/etc/apt/sources.list"
ShellInAnIcon_add_repokey_deb

[ShellInAnIcon_add_rep_deb]
set -x
export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
export DEBIAN_FRONTEND=noninteractive
apt-get --yes --force-yes install software-properties-common
apt-get --yes --force-yes install python-software-properties
add-apt-repository '$newrepo$'
exit $?

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

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

SUSE

```
[Actions]
DefVar $newrepo$
```

```
set $newrepo$ = "http://download.opensuse.org/repositories/home:/uibmz:/opsi:/opsi40/openSUSE_13
    .1/home:uibmz:opsi:opsi40.repo"

ShellInAnIcon_add_opsi_repository_suse

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

CentOS / Redhat

```
[Actions]
DefVar $newrepo$

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

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

ShellInAnIcon_add_repo_redhat

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

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

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

4.6.5 Delete a repository

Ubuntu / Debian

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

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

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

[PatchTextFile_del_repo_deb]
FindLine_StartingWith "$delrepo$"
DeleteTheLine
```

SUSE

```
[Actions]
DefVar $delrepo$

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

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

CentOS / Redhat

```
[Actions]
DefVar $delrepo$

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

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

4.6.6 Installing a package

Ubuntu / Debian

```
[Actions]

ShellInAnIcon_install_wget_debian

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

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

SUSE

```
[Actions]

ShellInAnIcon_install_wget_suse

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

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

CentOS / Redhat

```
[Actions]

ShellInAnIcon_install_wget_redhat

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

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

4.6.7 Integrated Example

```
[Actions]

DefVar $OS$
DefVar $distro_type$

DefStringlist $list$


set $OS$ = GetOS

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

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

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

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

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

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

Chapter 5

Start and Command Line Options

Since version 4.11.3, the *opsi-winst/opsi-script* program contains a manifest with the statement: `<requestedExecutionLevel level="highestAvailable" />`. This means that if *opsi-winst/opsi-script* is called on an NT6 OS by an Administrator, then it will run as an *elevated* process. If *opsi-winst/opsi-script* is called with normal user privileges, then it will run with the privileges of this user.

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

There are the following syntactical schemata:

(1) Show usage:

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

(2) Execute a script

`opsi-winst` <script file>
[`/logfile` <log file>]
[`/batch` | `/histolist` <opsi-winst config file path>]
[`/usercontext` <[domain\]user name>]
[`/parameter` parameter string]

(3) Execute a list of scripts (separated by semicolons) one by one:

`opsi-winst /scriptfile` <scriptfile> [;<script file>]* [`/logfile` <log file>]
[`/batch` | `/silent`]
[`/usercontext` <[domain\]user name>]
[`/parameter` <parameter string>]

4) Read the PC configuration from the opsi service and act accordingly, since *opsi-winst/opsi-script* 4.11.2

`opsi-winst /opsiservice` <opsiserviceurl>
`/clientid` <clientname>
`/username` <username>
`/password` <password>
[`/sessionid` <sessionid>]
[`/usercontext` <[domain\]username>]
[`/allloginscripts` | `/loginscripts`]
[`/silent`]

Note
At Linux the parameter delimiter is not "/" but "-". So instead of using `opsi-winst /help` like you would do at Windows you should use at Linux `opsi-script -help`.

Some explanations:

- Default name of the log file is an Windows `c:\opsi.org\log\opsi-script.log`
- The parameter string, which is marked by the option `/parameter`, is accessible for every called *opsi-winst/opsi-script* script (via the string function `ParamStr`).

Explanations to (2) and (3) :

- If option `/batch` is used, then *opsi-winst/opsi-script* shows only its "batch surface" offering no user dialogs. By option `/silent` event the batch surface is suppressed. Without using option `/batch` we get into the interactive mode where script file and log file can be chosen interactively (mainly for testing purposes).
- The `winstconfigfilepath` parameter which is designated by `/histofile` refers to a file in ini file format that holds the (in interactive mode) last used script file names. The dialogue surface presents a list box that presents these file names for choosing the next file to interpret. If `winstconfigfilepath` ends with "\" it is assumed to be a directory name and `WINST.INI` serves as file name.

Explanations to (4):

- The default for `clientid` is the full qualified computer name
- When called with option `/allloginscripts` or `/loginscripts` *opsi-winst/opsi-script* can do configurations for the logged in user (particularly in a Roaming Profile context). This is a cofunding feature - you need to buy it in order to use it.
  See at the opsi-manual for more information about *User Profile Management*.
- By option `/silent` the batch surface is suppressed.

The not interactive mode is implied.

5.1 Log File and Paths

The default log file name is `opsi-script.log`. You may find up to 8 Backup copys of old log files: from `opsi-script_0.log` until `opsi-script_8.log`.

The log file encoding is UTF-8.

By default log files are written at Windows into the directory `c:\opsi.org\log` which *opsi-winst/opsi-script* tries to create. If *opsi-winst/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-winst/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.

In addition, *opsi-winst/opsi-script* uses the logging directory for saving certain temporary files.

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 it self 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 be 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 imlemented 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]

- `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 wthout changing the script. Default: *0*
  see also [SetLogLevel]

- `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]

Chapter 6

Additional Configurations

6.1 Central Logging of Error Messages

If the *opsi-winst/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-winst/opsi-script* [W/L]

Note
For Linux see: Section 4.3

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

Since version 4.11.3.5 the *opsi-winst* searches the skin directory in the following order (directory with the first skin.ini to be found wins):

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

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

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.

`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 and includes as well. You have to give the command:
`encoding=`<encoding>
This command can be at any position in the code (Even before [actions]). As <encoding> you may give one of the following values:

Table 6.1: Encodings

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

Sources see:
https://en.wikipedia.org/wiki/Code_page

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

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

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

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

Chapter 7

The *opsi-winst/opsi-script* Script

On principle: *opsi-winst/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 *opsi-winst/opsi-script* script. The purpose is to identify the book marks of a script: in which way we to 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

opsi-winst/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 *opsi-winst/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_win7
else
  if ( $MSVersion$ = "5.1" )
    sub_install_winXP
  else
    stop "not a supported OS-Version"
  endif
endif


[sub_install_win7]
Files_copy_win7
WinBatch_Setup

[sub_install_winXP]
```

```
Files_copy_XP
WinBatch_SetupXP

[Files_copy_win7]
copy "%scriptpath%\files_win7\*.*" "c:\temp\installation"

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

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

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

How can we read the sections of this script?

7.2 Primary and Secondary Subprograms of a *opsi-winst/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 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 *opsi-winst/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.

Note
If a script is run as *userLoginScript* and it contains a section [ProfileActions], so the script interpretation will be started at the `ProfleActions` section.

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

The primary or general control sections comprise

- the **Actions** section
- **Sub** sections (0 to n subroutines called by the **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,
- LinkFolder sections,
- opsiServiceCall sections,
- ExecPython sections,
- ExecWith sections,
- LDAPsearch sections.

Meaning and syntax of the different section types are treated in Chapter 9 and Chapter 10.

7.3 String Expressions in a *opsi-winst/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 Section 9.4)

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. Lets have a closer look at the variables and constants of a *opsi-winst/opsi-script* script:

Chapter 8

Definition and Use of Variables and Constants in a *opsi-winst/opsi-script* Script

8.1 General

In a *opsi-winst/opsi-script* script, variables and constants appear as "words", that are interpreted by *opsi-winst/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-winst/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-winst/opsi-script* program and cannot be changed in a script. Before interpreting the script *opsi-winst/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-winst/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-winst/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-winst/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:

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

At this moment the following constants are implemented:

8.2.3 System paths

8.2.3.1 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:\Documents and Settings*

8.2.3.2 Common (AllUsers) directories [W]

`%AllUsersProfileDir%` or `%CommonProfileDir%` : *c:\Documents and Settings\All Users*

`%CommonStartMenuPath%` or `%CommonStartmenuDir%` : *c:\Documents and Settings\All Users\Startmenu*

`%CommonAppdataDir%` : *c:\Documents and Settings\All Users\Application Data*

`%CommonDesktopDir%`

`%CommonStartupDir%`

`%CommonProgramsDir%`

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

8.2.3.3 Default User Directory [W]

`%DefaultUserProfileDir%`

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

`%AppdataDir%` or `%CurrentAppdataDir%` : //since 4.10.8.13
NT5: *c:\Documents and Settings\%USERNAME%\Application Data* NT6: *c:\users\%USERNAME%\Appdata\Roaming*

`%CurrentStartmenuDir%`

`%CurrentDesktopDir%`

`%CurrentStartupDir%`

`%CurrentProgramsDir%`

`%CurrentSendToDir%`

`%CurrentProfileDir%` //since 4.11.2.1

8.2.3.5 /AllNtUserProfiles directory constants [W]

In *Files* sections that are called with option `/AllNtUserProfiles` 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.

`%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%`
NT5: *c:\Documents and Settings\%USERNAME%*
NT6: *c:\users\%USERNAME%*

8.2.4 *opsi-winst/opsi-script* Path and Directory [W/L]

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

`%ScriptPath%` or `%ScriptDir%` : represents the path of the current *opsi-winst/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.

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

`%WinstDir%` : The location (without closing backslash) of the running *opsi-winst/opsi-script*.

`%WinstVersion%` : Version string of the running *opsi-winst/opsi-script*.

`%Logfile%` : The name of the logfile which *opsi-winst/opsi-script* is using.

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

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

`%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:

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

results to the following log:

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

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

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

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

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

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

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

Else
EndIf
```

8.2.5 Network Information [W/L]

`%Host%` : (Deprecated) 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%` : 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]

`%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 usally the clint FQDN used by opsi.

`%opsiservicePassword%` : The user password used for the connection to the opsi service. The password is eliminated when logging by the standard *opsi-winst/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 combinated 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%` : (Deprecated) 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]

8.3.1 Declaration

String variables must be declared before they can be used. The syntax for the declaration reads

`DefVar` <variable name>

e.g.

```
DefVar $MsVersion$
```

Explanation:

- Variable names do not necessarily start or end with a dollar sign, but this is recommended as a convention to understand their functioning 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` <Variablenname> `=` <Value>

<Value> means any (String valued) expression.

Examples (For Examples see Section 9.4):

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

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

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

8.3.3 Use of variables in String expressions

In primary sections of a *opsi-winst/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-winst/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 $Home$
DevVar $MailLocation$
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]

Variables for string lists must be declared in a DefStringList statement, e.g.

```
DefStringList SMBMounts
```

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 Section 9.5).

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

Chapter 9

Syntax and Meaning of Primary Sections of a *opsi-winst/opsi-script* Script [W/L]

As shortly presented in chapter 4 the Actions section of a script can be regarded as a the main method of the *opsi-winst/opsi-script* 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 *opsi-winst/opsi-script* script.

9.1 Primary Sections [W/L]

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 *opsi-winst/opsi-script* 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]`

Note
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. Section 9.16).

Caution
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 opsi-manual and Section 9.11.

9.2 Parametrizing *opsi-winst/opsi-script* [W/L]

9.2.1 Specification of Logging Level [W/L]

Caution
The old function `LogLevel=` is deprecated since *opsi-winst/opsi-script* 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. section 6.3). In the second case, *opsi-winst/opsi-script* tries to evaluate the string expression as a number. There exist ten levels from 0 up to 9.

```
0 = nothing (absolute nothing)
1 = essential ("essential information")
2 = critical (unexpected errors that my cause a program abort)
3 = error (Errors that don't will abort the running program)
4 = warning (you should have a look at this)
5 = notice (Important statements to the program flow)
6 = info (Additional Infos)
7 = debug (important debug messages)
8 = debug2 (a lot more debug informations and data)
9 = confidential (passwords and other security relevant data)
```

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 out put is not assinged 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 assinged 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-winst Version [W/L]

The statement

`requiredWinstVersion` <RELATION SYMBOL> <NUMBER STRING>

e.g.

```
requiredWinstVersion >= "4.3"
```

makes *opsi-winst/opsi-script* check if the desired version state is given. Otherwise an error message windows pops up.

This feature exists since *opsi-winst/opsi-script* version 4.3. For an earlier version, the statement is unknown, and the statement itsself is a syntactical error which will be indicated by syntax error window (cf. the following section). Therefore the statement can be used independently of the currently used *opsi-winst/opsi-script* version as long as the required version is at least version 4.3.

9.2.3 Reacting on Errors [W/L]

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

1. illegal statements which cannot be interpreted by *opsi-winst/opsi-script* (syntactical errors),
2. failing statements which cannot be executed because of external, objective reasons (execution errors).

In principal, syntactical errors are indicated by a pop up window for immediate correction, execution errors are logged in a log file to be analysed later.

The behaviour of *opsi-winst/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-winst/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.

Caution
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-winst/opsi-script* keeps staying on top.

9.3 Show window mode / Skin / Activity [W/L]

- `SetSkinDirectory` <skindir> // [W/L]
  Sets the skin directory to use and loads the skin. If this command is used wit an empty or invalid path, the default skin dir is used. The default skin dir `%WinstDir%\winstskin`.

Example:

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

To change the modes of how the *opsi-winst/opsi-script* window is displayed, use these commands:

- `NormalizeWinst`
  Sets the *opsi-winst/opsi-script* window to the *normal* mode
- `IconizeWinst`
  Sets the *opsi-winst/opsi-script* window to the *minimized* mode
- `MaximizeWinst` //since 4.11.5
  Sets the *opsi-winst/opsi-script* window to the *maximized* mode

- `RestoreWinst`
  Sets the *opsi-winst/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.

9.4 String Expressions, String Values, and String Functions [W/L]

A String expression can be

- an elementary String value
- a nested String value
- a String variable
- the concatenation of other String expressions
- a String valued function call

9.4.1 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:

```
DefVar $ExampleString$
Set $ExampleString$ = "my Text"
```

9.4.2 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:

```
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:

```
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: `Set $citation$ = 'he said "Yes"'`

9.4.3 String Concatenation

String concatenation is written using the addition sign ("+")

<String expression> + <String expression>

Example:

```
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".

9.4.4 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. Section 8.3.

The following sections present the variety of string functions.

9.4.5 String Functions which Return the OS Type

- `GetOS : string` [W/L]
  The function tells which type of operating system is running.
  We recommend to use `GetMsVersionInfo`.
  GetOS` returns one of the following values:
  "Windows_NT" (including Windows 2000 to Windows 10)
  "Linux"

- `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 systems of type Windows NT the Microsoft version info as indicated by the API, e.g. a Windows XP system produces the result
  "5.1"

Table 9.1: Windows Versions

GetMsVersionInfo	**Windows Version**
5.0	Windows 2000
5.1	Windows XP (Home, Prof)
5.2	XP 64 Bit, 2003, Home Server, 2003 R2
6.0	Vista, 2008
6.1	Windows 7, 2008 R2
6.2	Windows 8, 2012
6.3	Windows 8.1, 2012 R2
10.0	Windows 10

see also `GetMsVersionMap`

- `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]

- `GetSystemType : string` [W/L]
  checks the installed Windows OS if it can be assumed that the system is 64 Bit. In this case the value is *64 Bit System* otherwise *x86 System*.

9.4.6 String Functions for Retrieving Environment or Command Line Data [W/L]

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 The function passes the the parameter string of the *opsi-winst/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 returns the exit code (also called ErroLevel) of the last Winbatch call. GetUserSID(<Windows Username>) returns the SID for a given user (possibly with domain prefix in the form DOMAIN\USER).

- `EnvVar (<environment variable>) : string` [W/L] [W/L]
  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]
  The function passes the the parameter string of the *opsi-winst/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` [W/L]
  returns the exit code (also called ErroLevel) of the last Winbatch call.

- `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-winst/opsi-script* by the optional parameter `/usercontext`. IF this parameter was not userd the returned string is empty.

9.4.7 Reading Values from the Windows Registry and Transforming Values into Registry Format [W]

- `getRegistryValue` (<keystr>, <varstr> [, <access str>]) : 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:

```
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` [W]
  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` .
  see also : [getRegistryValue]

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\\"

9.4.8 Reading Values from ini files [W/L]

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]
  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 deprecated. The syntax reads:

- `GetIni ( <Stringausdruck> [ <character sequence> ] <character sequence> )`
  (Deprecated) The <String expression> is interpreted as file name, the first <character sequence> as section name, the second as key name.

9.4.9 Reading Product Properties [W/L]

- `GetProductProperty` (<PropertyName>, <DefaultValue>)
  where $PropertyName$ and $DefaultValue$ are String expressions. If *opsi-winst/opsi-script* is connected to the opsi configuration service the product property is retrieved from the service. If the *opsi-winst/opsi-script* is not connected to the service or for other reasons the the call fails, the given `<DefaultValue>` will be 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")
```

- `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`
- `IniVar`(<PropertyName>)
  (deprecated: use GetProductProperty)

9.4.10 Retrieving Data from etc/hosts [W/L]

- `GetHostsName`(<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.

9.4.11 String Handling [W/L]

- `ExtractFilePath` (<path>) : `string` [W/L]
  interprets the passed String value as file or path name and returns the path part (the string up to the last "\", including it).
- `StringSplit (`STRINGWERT1, STRINGWERT2, INDEX)``
  (deprecated: use `splitString` / `takeString`)
  see also : [splitString]
  see also : [takeString]

- `takeString` (<index>, <list>) : `string` [W/L]
  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 Section 9.5).
  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 slpitted 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>`)`
  (deprecated: 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> : string //since 4.12.0.13 [W/L]
  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.
- `takeFirstStringContaining(`<list>,<search string>`) : string` [W/L]
  returns the first string from <list> which contains <search string>.
  Retruns a empty string if no matching string is found.
  see also : [takeFirstStringContaining]
- `trim(`<string>`) : string` [W/L]
  cuts leading and trailing white space from <string>.
- `lower(`<string>`) : string` [W/L]
  returns <string> with lower case.
- `upper(`<string>`)` [W/L]
  returns <string> with upper case.
- `contains(`<str>, <substr>`) : bool` //since 4.11.3: true if <substr> in <str> [W/L]
  A boolean function which returns *true* if <str> contains <substr>. This function is case sensitive.
  Available since 4.11.3
  Example:

```
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]
  returns a string, which has all occurrences of <oldPattern> replaced with <newPattern> given then input string <string>.

Example:

```
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]
  Returns the number of chars in in <string>

Example:

```
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 (numner)` //since 4.11.3 [W/L]
  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:

```
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]
  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:

```
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]
  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'", "'")
comment "compare values"
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
comment "double quote"
set $ConstTest$ = "b"
set $CompValue$ = unquote('"b"', '"')
comment "compare values"
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
comment "quote string will be trimmed and then only the first char is used"
comment "note: brackets are different chars"
set $ConstTest$ = "b]"
set $CompValue$ = unquote("[b]", " [{ ")
comment "compare values"
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
comment "not usable to remove brackets"
set $ConstTest$ = "b]"
set $CompValue$ = unquote("[b]", "[")
set $CompValue$ = unquote($CompValue$,"]")
set $CompValue$ = unquote("[b]", "]")
set $CompValue$ = unquote($CompValue$,"[")
set $CompValue$ = unquote(unquote("[b]", "["),"]")
comment "compare values"
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
comment "if string not quoted it will be come back without changes"
set $ConstTest$ = "b"
set $CompValue$ = unquote("b", "'")
comment "compare values"
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
```

```
endif
```

- `unquote2(<string>,<quote-string>) : string` //since 4.11.5.2 [W/L]
  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.

- `HexStrToDecStr (<hexstring>) : string` [W/L]
  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]
  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.

```
message "DecStrToHexStr"
set $ConstTest$ = "0407"
set $tmp$ = "1031"
set $CompValue$ = DecStrToHexStr($tmp$,"4")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

message "DecStrToHexStr"
set $ConstTest$ = "407"
set $tmp$ = "1031"
set $CompValue$ = DecStrToHexStr($tmp$,"2")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `base64EncodeStr(<string>) : string` [W/L]
  returns the base64 encoded value of <string>.

- `base64DecodeStr(<string>) : string` [W/L]
  returns the base64 decoded value of <string>.

```
message "base64EncodeStr"
set $ConstTest$ = "YWJjZGVm"
set $tmp$ = "abcdef"
set $CompValue$ = base64EncodeStr($tmp$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
```

```
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

comment ""
comment "------------------------------"
comment "Testing: "
message "base64DecodeStr"
set $ConstTest$ = "abcdef"
set $tmp$ = "YWJjZGVm"
set $CompValue$ = base64DecodeStr($tmp$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `encryptStringBlow(`<keystring>,<datastring>`) : string` [W/L]
  Encrypted <datastring> with the Key <keystring> under application of Blowfish and returns the encrypted value.

- `decryptStringBlow(`<keystring>,<datastring>`) : string` [W/L]
  Decrypts <datastring> with the Key <keystring> under the application of Blowfish and returns the decrypted value.

```
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]
  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]
  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 Section 6.3.

- `strLoadTextFile (`<file name>`) : string` [W/L]
  returns the first line of <filename> as String.
  see also : [loadTextFile]
  see also : [strLoadTextFileWithEncoding]
  see also : [loadUnicodeTextFile]
  see also : [loadTextFileWithEncoding]

- `strLoadTextFileWithEncoding (` <filename> , <encoding>`) : string` [W/L]
  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 : Section 6.3

- `GetShortWinPathName(<longpath string>)` //since 4.11.5.2 [W]
  Returns the short path (8.3) from <longpath string>. If ther is no short path for <longpath string>, so you will get an empty string.
  Example: `GetShortWinPathName("C:\Program Files (x86)")` returns `"C:\PROGRA~2"`

9.4.12 Other String Functions

- `RandomStr : string` [W/L]
  returns a random String of length 10 where upper case letters, lower case letters and digits are mixed (for creating passwords). More exactly: 2 lower case chars, 2 upper case chars, 2 special chars and 4 digits. The possible special chars are:
  !,$,(,),,+,/,;,=,?,[,],{,},ß,~,§,°*

- `CompareDotSeparatedNumbers(`<string1>, <string2>`) : string` [W/L]
  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]

Example:
The Code:

```
        comment "Testing: "
        message "CompareDotSeparatedNumbers"
        set $string1$ = "1.2.3.4.5"
        set $string2$ = "1.2.3.4.5"
        set $ConstTest$ = "0"
        set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is equal to "+$string2$
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif

        set $string1$ = "1.2.31.4.5"
        set $string2$ = "1.2.13.4.5"
        set $ConstTest$ = "1"
        set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is higher then "+$string2$
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif

        set $string1$ = "1.2.3.4.5"
        set $string2$ = "1.2.13.4.5"
        set $ConstTest$ = "-1"
        set $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is lower then "+$string2$
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif

        comment ""
        comment "------------------------------"
        comment "Testing: "
        message "CompareDotSeparatedStrings"
        set $string1$ = "1.a.b.c.3"
        set $string2$ = "1.a.b.c.3"
        set $ConstTest$ = "0"
        set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is equal to "+$string2$
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif
```

leads to the following log:

```
comment: Testing:
message CompareDotSeparatedNumbers

Set  $string1$ = "1.2.3.4.5"
  The value of the variable "$string1$" is now: "1.2.3.4.5"

Set  $string2$ = "1.2.3.4.5"
  The value of the variable "$string2$" is now: "1.2.3.4.5"

Set  $ConstTest$ = "0"
  The value of the variable "$ConstTest$" is now: "0"

Set  $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "0"

If
```

```
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.2.3.4.5 is equal to 1.2.3.4.5

Else
EndIf

Set  $string1$ = "1.2.31.4.5"
  The value of the variable "$string1$" is now: "1.2.31.4.5"

Set  $string2$ = "1.2.13.4.5"
  The value of the variable "$string2$" is now: "1.2.13.4.5"

Set  $ConstTest$ = "1"
  The value of the variable "$ConstTest$" is now: "1"

Set  $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "1"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.2.31.4.5 is higher then 1.2.13.4.5

Else
EndIf

Set  $string1$ = "1.2.3.4.5"
  The value of the variable "$string1$" is now: "1.2.3.4.5"

Set  $string2$ = "1.2.13.4.5"
  The value of the variable "$string2$" is now: "1.2.13.4.5"

Set  $ConstTest$ = "-1"
  The value of the variable "$ConstTest$" is now: "-1"

Set  $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "-1"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.2.3.4.5 is lower then 1.2.13.4.5

Else
EndIf
```

- `CompareDotSeparatedStrings(`<string1>, <string2>`) : string` [W/L]
  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]

Example:
The Code:

```
        comment "Testing: "
        message "CompareDotSeparatedStrings"
        set $string1$ = "1.a.b.c.3"
        set $string2$ = "1.a.b.c.3"
        set $ConstTest$ = "0"
        set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is equal to "+$string2$
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif

        set $string1$ = "1.a.b.c.3"
        set $string2$ = "1.A.B.C.3"
        set $ConstTest$ = "0"
        set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is equal to "+$string2$
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif

        set $string1$ = "1.a.cb.c.3"
        set $string2$ = "1.a.b.c.3"
        set $ConstTest$ = "1"
        set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is higher then "+$string2$
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif

        set $string1$ = "1.a.ab.c.3"
        set $string2$ = "1.a.b.c.3"
        set $ConstTest$ = "-1"
        set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is lower then "+$string2$
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif

        set $string1$ = "1.2.13.4.5"
        set $string2$ = "1.2.3.4.5"
        set $ConstTest$ = "-1"
        set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is lower then "+$string2$
                comment "using CompareDotSeparatedStrings give wrong results on numbers"
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif

        set $string1$ = "1.2.3.4.5"
        set $string2$ = "1.2.13.4.5"
        set $ConstTest$ = "1"
        set $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
        if ($ConstTest$ = $CompValue$)
                comment "passed"
                comment $string1$+" is higher then "+$string2$
                comment "using CompareDotSeparatedStrings give wrong results on numbers"
        else
                set $TestResult$ = "not o.k."
                LogWarning "failed"
        endif
```

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"
  The value of the variable "$ConstTest$" is now: "0"

Set  $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "0"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.a.b.c.3 is equal to 1.a.b.c.3

Else
EndIf

Set  $string1$ = "1.a.b.c.3"
  The value of the variable "$string1$" is now: "1.a.b.c.3"

Set  $string2$ = "1.A.B.C.3"
  The value of the variable "$string2$" is now: "1.A.B.C.3"
```

```
Set  $ConstTest$ = "0"
  The value of the variable "$ConstTest$" is now: "0"

Set  $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "0"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.a.b.c.3 is equal to 1.A.B.C.3

Else
EndIf

Set  $string1$ = "1.a.cb.c.3"
  The value of the variable "$string1$" is now: "1.a.cb.c.3"

Set  $string2$ = "1.a.b.c.3"
  The value of the variable "$string2$" is now: "1.a.b.c.3"

Set  $ConstTest$ = "1"
  The value of the variable "$ConstTest$" is now: "1"

Set  $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "1"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.a.cb.c.3 is higher then 1.a.b.c.3

Else
EndIf

Set  $string1$ = "1.a.ab.c.3"
  The value of the variable "$string1$" is now: "1.a.ab.c.3"

Set  $string2$ = "1.a.b.c.3"
  The value of the variable "$string2$" is now: "1.a.b.c.3"

Set  $ConstTest$ = "-1"
  The value of the variable "$ConstTest$" is now: "-1"

Set  $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "-1"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.a.ab.c.3 is lower then 1.a.b.c.3
```

```
Else
EndIf

Set  $string1$ = "1.2.13.4.5"
  The value of the variable "$string1$" is now: "1.2.13.4.5"

Set  $string2$ = "1.2.3.4.5"
  The value of the variable "$string2$" is now: "1.2.3.4.5"

Set  $ConstTest$ = "-1"
  The value of the variable "$ConstTest$" is now: "-1"

Set  $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "-1"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.2.13.4.5 is lower then 1.2.3.4.5
  comment: using CompareDotSeparatedStrings give wrong results on numbers

Else
EndIf

Set  $string1$ = "1.2.3.4.5"
  The value of the variable "$string1$" is now: "1.2.3.4.5"

Set  $string2$ = "1.2.13.4.5"
  The value of the variable "$string2$" is now: "1.2.13.4.5"

Set  $ConstTest$ = "1"
  The value of the variable "$ConstTest$" is now: "1"

Set  $CompValue$ = CompareDotSeparatedStrings($string1$, $string2$)
  The value of the variable "$CompValue$" is now: "1"

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
  comment: 1.2.3.4.5 is higher then 1.2.13.4.5
  comment: using CompareDotSeparatedStrings give wrong results on numbers

Else
EndIf
```

- `getDiffTimeSec` [W/L]
  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]

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 claculate 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]
  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 $CompValue$ = GetMyIpByTarget("%opsiServer%")
```

see also : [GetIpByName]
see also : [IPAddress]

* `GetIpByName(`<ip addr / ip name>`)` [W/L]
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]

- `getLastExitCode : string (exitcode)` [W/L]
  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:

```
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%
```

9.4.13 (String-) Functions for Licence Management [W/L]

- `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.

9.4.14 Retrieving Error Infos from Service Calls [W/L]

- `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
```

9.5 String List Functions and String List Processing [W/L]

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. They are defined by the statement

`DefStringList` <VarName>

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 expressions. 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.

For the following examples we declare a String list variable *$list1$*:

```
DefStringList $list1$
```

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 *KEY=VALUE*. In fact, each map should establish a function which associates a *VALUE* to a *KEY*, and any *KEY* should occur at most once as the first part of a line (whereas different *KEY's may be associated with identical 'VALUE* parts).

9.5.1 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.UUID=303032343144323730434336FFFFFFFF
sysinfo.SKU Number=
sysinfo.Family=
board.Manufacturer=Gigabyte Technology Co., Ltd.
board.Product=GA-MA78GM-UD2H
board.Version=x.x
board.Serial Number=
board.Asset Tag=
board.Feature Flags=01101001
board.Location in Chassis=
board.Chassis Handle=6261
board.Board Type=79 Unknown
board.Number of Contained Object Handles=116
enclosure.Manufacturer=Gigabyte Technology Co., Ltd.
enclosure.Version=
enclosure.Serial Number=
enclosure.Asset Tag Number=
enclosure.Type=Desktop
enclosure.Power Supply State=Unknown
enclosure.BootUp State=Unknown
```

- `getLinuxVersionMap : stringlist` //since 4.11.4 [L]
  get OS information and writes them to hash map string list.
  There are the folowing 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
SubRelease
```

- `GetMSVersionMap : stringlist` [W]
  get OS information and writes them to hash map string list.
  There are the folowing 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

```
0x0000001 (VER_NT_WORKSTATION)
0x0000002 (VER_NT_DOMAIN_CONTROLLER)
0x0000003 (VER_NT_SERVER)
```

SuiteMask

```
0x00000001 (VER_SUITE_SMALLBUSINESS)
0x00000002 (VER_SUITE_ENTERPRISE)
0x00000004 (VER_SUITE_BACKOFFICE)
0x00000008 (VER_SUITE_COMMUNICATIONS)
0x00000010 (VER_SUITE_TERMINAL)
0x00000020 (VER_SUITE_SMALLBUSINESS_RESTRICTED)
0x00000040 (VER_SUITE_EMBEDDEDNT)
0x00000080 (VER_SUITE_DATACENTER)
0x00000100 (VER_SUITE_SINGLEUSERTS)
0x00000200 (VER_SUITE_PERSONAL)
0x00000400 (VER_SUITE_SERVERAPPLIANCE)
```

- `ReleaseID` 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

DecNum	HexNum	Text
00	00	An unknown product
01	01	Ultimate Edition"
02	02	Home Basic Edition
03	03	Home Premium Edition
04	04	Enterprise Edition
05	05	Home Basic Edition
06	06	Business Edition
07	07	Server Standard Edition (full installation)
08	08	Server Datacenter Edition (full installation)
09	09	Small Business Server
10	0A	Server Enterprise Edition (full installation)
11	0B	Starter Edition
12	0C	Server Datacenter Edition (core installation)
13	0D	Server Standard Edition (core installation)
14	0E	Server Enterprise Edition (core installation)
15	0F	Server Enterprise Edition for Itanium-based Systems
16	10	Business Edition
17	11	Web Server Edition (full installation)
18	12	Cluster Server Edition
19	13	Home Server Edition
20	14	Storage Server Express Edition
21	15	Storage Server Standard Edition

22	16	Storage Server Workgroup Edition
23	17	Storage Server Enterprise Edition
24	18	Server for Small Business Edition
25	19	Small Business Server Premium Edition
26	1A	PRODUCT_HOME_PREMIUM_N
27	1B	PRODUCT_ENTERPRISE_N
28	1C	PRODUCT_ULTIMATE_N
29	1D	PRODUCT_WEB_SERVER_CORE
30	1E	Windows Essential Business Server Management Server
31	1F	Windows Essential Business Server Security Server
32	20	Windows Essential Business Server Messaging Server
33	21	Server Foundation
34	22	PRODUCT_HOME_PREMIUM_SERVER
35	23	PRODUCT_SERVER_FOR_SMALLBUSINESS_V
36	24	Server Standard Edition without Hyper-V (full installation)
37	25	Server Datacenter Edition without Hyper-V (full installation)
38	26	Server Enterprise Edition without Hyper-V (full installation)
39	27	Server Datacenter Edition without Hyper-V (core installation)
40	28	Server Standard Edition without Hyper-V (core installation)
41	29	Server Enterprise Edition without Hyper-V (core installation)
48	30	PRODUCT_PROFESSIONAL
49	31	PRODUCT_PROFESSIONAL_N
50	32	PRODUCT_SB_SOLUTION_SERVER
51	33	PRODUCT_SERVER_FOR_SB_SOLUTIONS
52	34	PRODUCT_STANDARD_SERVER_SOLUTIONS
53	35	PRODUCT_STANDARD_SERVER_SOLUTIONS_CORE
54	36	PRODUCT_SB_SOLUTION_SERVER_EM
55	37	PRODUCT_SERVER_FOR_SB_SOLUTIONS_EM
56	38	PRODUCT_SOLUTION_EMBEDDEDSERVER
57	39	PRODUCT_SOLUTION_EMBEDDEDSERVER_CORE
59	3B	PRODUCT_ESSENTIALBUSINESS_SERVER_MGMT
60	3C	PRODUCT_ESSENTIALBUSINESS_SERVER_ADDL
61	3D	PRODUCT_ESSENTIALBUSINESS_SERVER_MGMTSVC
62	3E	PRODUCT_ESSENTIALBUSINESS_SERVER_ADDLSVC
63	3F	PRODUCT_SMALLBUSINESS_SERVER_PREMIUM_CORE
64	40	PRODUCT_CLUSTER_SERVER_V
65	41	PRODUCT_EMBEDDED
66	42	PRODUCT_STARTER_E
67	43	PRODUCT_HOME_BASIC_E
68	44	PRODUCT_HOME_PREMIUM_E
69	45	PRODUCT_PROFESSIONAL_E
70	46	PRODUCT_ENTERPRISE_E
71	47	PRODUCT_ULTIMATE_E
72	48	PRODUCT_ENTERPRISE_EVALUATION
84	54	PRODUCT_ENTERPRISE_N_EVALUATION
98	62	PRODUCT_CORE_N
99	63	PRODUCT_CORE_COUNTRYSPECIFIC
100	64	PRODUCT_CORE_SINGLELANGUAGE
101	65	PRODUCT_CORE
121	79	PRODUCT_EDUCATION
122	7A	PRODUCT_EDUCATION_N
125	7D	Windows Enterprise 2015 LTSB
126	7E	Windows Enterprise 2015 LTSB N
129	81	Windows Enterprise 2015 LTSB Evaluation
130	82	Windows Enterprise 2015 LTSB N Evaluation

Example:
The Code

```
DefStringList $INST_Resultlist$
DefStringList $INST_Resultlist2$

message "getMSVersionMap"
comment "get value by winst function"
set $INST_Resultlist$ = getMSVersionMap
```

produces the following log:

```
message getMSVersionMap
comment: get value by winst function

Set  $INST_Resultlist$ = getMSVersionMap
    retrieving strings from getMSVersionMap [switch to loglevel 7 for debugging]
        (string   0)major_version=5
        (string   1)minor_version=1
        (string   2)build_number=2600
        (string   3)platform_id=2
        (string   4)csd_version=Service Pack 3
        (string   5)service_pack_major=3
        (string   6)service_pack_minor=0
        (string   7)suite_mask=256
        (string   8)product_type_nr=1
        (string   9)2003r2=false
```

Note
Background infos for getMSVersionMap

- http://msdn.microsoft.com/en-us/library/ms724385%28VS.85%29.aspx
- http://msdn.microsoft.com/en-us/library/dd419805.aspx
- http://msdn.microsoft.com/en-us/library/ms724833%28VS.85%29.aspx

- `getFileInfoMap(` <file name> `)` : `stringlist` [W]
- `getFileInfoMap32(` <file name> `)` : `stringlist` //since 4.11.6.6 [W]
- `getFileInfoMap64(` <file name> `)` : `stringlist` //since 4.11.6.6 [W]
- `getFileInfoMapSynative(` <file name> `)` : `stringlist` //since 4.11.6.6 [W]

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

```
DefStringList FileInfo
DefVar $InterestingFile$
Set $InterestingFile$ = "c:\program files\my program.exe"
set FileInfo = getFileInfoMap($InterestingFile$)
```

we get the value associated with key "FileVersion" from the call

```
DefVar $result$
set $result$ = getValue("FileVersion", FileInfo)
```

(for the function getValue cf. Section 9.5.4).

Example:
The code:

```
set $InterestingFile$ = "%winstdir%\winst.exe"
if not (FileExists($InterestingFile$))
        set $InterestingFile$ = "%winstdir%\winst32.exe"
endif
set $INST_Resultlist$ = 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-winst
        (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:

```
message "Locale Infos"
set $INST_Resultlist$ = GetLocaleInfoMap
```

produces e.g the log:

```
message Locale Infos

Set $INST_Resultlist$ = GetLocaleInfoMap
    retrieving strings from GetLocaleInfoMap [switch to loglevel 7 for debugging]
        (string   0)language_id_2chars=DE
        (string   1)language_id=DEU
        (string   2)localized_name_of_language=Deutsch (Deutschland)
        (string   3)English_name_of_language=German
        (string   4)abbreviated_language_name=DEU
        (string   5)native_name_of_language=Deutsch
        (string   6)country_code=49
        (string   7)localized_name_of_country=Deutschland
        (string   8)English_name_of_country=Germany
        (string   9)abbreviated_country_name=DEU
        (string  10)native_name_of_country=Deutschland
        (string  11)default_language_id=0407
        (string  12)default_language_id_decimal=1031
        (string  13)default_country_code=49
        (string  14)default_oem_code_page=850
        (string  15)default_ansi_code_page=1252
        (string  16)default_mac_code_page=10000
        (string  17)system_default_language_id=0407
        (string  18)system_default_posix=de_DE
        (string  19)system_default_lang_region=de-DE
```

Usage: If we define and call

```
DefStringList $languageInfo$
set  $languageInfo$ = GetLocaleInfoMap
```

we get the value associated with key "language_id_2chars" from the call

```
DefVar $result$
set $result$ = getValue("language_id_2chars", $languageInfo$)
```

(for the function getValue cf. Section 9.5.4). We may now write scripts using a construct like

```
if getValue("language_id_2chars", languageInfo) = "DE"
   ; installiere deutsche Version
else
   if getValue("language_id_2chars", languageInfo) = "EN"
   ; installiere englische Version
   endif
endif
```

Note
Background infos for GetLocaleInfoMap:

- http://msdn.microsoft.com/en-us/library/cc233968.aspx
- http://msdn.microsoft.com/en-us/library/0h88fahh.aspx
- bcp 47 validator:
  http://schneegans.de/lv/?tags=de-de-1996&format=text
- http://www.iana.org/assignments/language-subtag-registry
- http://www.the-localization-tool.com/?p=698

- `getLocaleInfo`
  (deprecated): use `GetLocaleInfoMap` .
  see also : [GetLocaleInfoMap]

- `getProductMap` // since 4.11.2.4 [W/L]
  returns a info map of the opsi product you are just installing.
  It works only if *opsi-winst/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:

```
set $INST_Resultlist$ = getProductMap
set $string1$ = getValue("id", $INST_Resultlist$)
```

produces e.g the log:

```
Set  $INST_Resultlist$ = getProductMap
    retrieving strings from getProductMap [switch to loglevel 7 for debugging]
        (string   0)id=opsi-script-test
        (string   1)name=opsi-winst test
        (string   2)description=Test  and example script for opsi-winst
        (string   3)advice=
        (string   4)productversion=4.11.2
        (string   5)packageversion=1
        (string   6)priority=0
        (string   7)installationstate=unknown
        (string   8)lastactionrequest=setup
        (string   9)lastactionresult=successful
        (string  10)installedversion=4.11.2
        (string  11)installedpackage=1
        (string  12)installedmodificationtime=
        (string  13)actionrequest=setup


Set  $string1$ = getValue("id", $INST_Resultlist$)
    retrieving strings from $INST_Resultlist$ [switch to loglevel 7 for debugging]
        (string   0)id=opsi-script-test
        (string   1)name=opsi-winst test
        (string   2)description=Test  and example script for opsi-winst
        (string   3)advice=
        (string   4)productversion=4.11.2
        (string   5)packageversion=1
        (string   6)priority=0
        (string   7)installationstate=unknown
        (string   8)lastactionrequest=setup
        (string   9)lastactionresult=successful
        (string  10)installedversion=4.11.2
        (string  11)installedpackage=1
        (string  12)installedmodificationtime=
        (string  13)actionrequest=setup

  The value of the variable "$string1$" is now: "opsi-script-test"
```

9.5.2 Producing String Lists from Strings [W/L]

- `createStringList` (<string0>, <string1> ,...) : `stringlist` [W/L]
  forms 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]
generates the list of partial strings of <string1> (including empty strings) before resp. between the occurences of <string2>. E.g.,

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

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.

- `splitStringOnWhiteSpace` (<string>) : `stringlist` [W/L]
  slices StringVal by the "white spots" in it. E. g.

```
set $list1$ = splitStringOnWhiteSpace("Status   Lokal     Remote          Netzwerk")
```

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.

9.5.3 Loading Lines of a Text File into a String List

- `loadTextFile` (<file name>) : `stringlist` [W/L]
  reads the file <file name> and generates the string list, that contains all lines of the file.

- `loadTextFileWithEncoding`(<file name> , <encoding>) : `stringlist` [W/L]
  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.

- `loadUnicodeTextFile (`<file name>`) : stringlist` [W]
  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.

- `getSectionNames(`<ini-file>`) : stringlist` [W/L]
  interprets the specified file as an inifile, looks for list of all lines of form
  [<SectionName>]
  and returns the pure section names (without brackets).

9.5.4 Simple String Values generated from String Lists or Files [W/L]

- `composeString (`<string list>, <Link>`) : string` [W/L]
  With this function, the elements of any String list can be glued to one another, mediated by a "glue string".
  E.g. if *$list1$* 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]
  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]
  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]
  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 9.5.1).

- `getValueBySeparator(`<key string>,<separator string>,<hash string list>` ) : string` //since 4.11.2.1 [W/L]
  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]
  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]
  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]
  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".

9.5.5 Producing String Lists from opsi-winst Sections [W/L]

- `retrieveSection (`<section name>`) : stringlist` [W/L]
  gives the lines of the specified section as string list.

- `getOutStreamFromSection (`<dos section name>`) : stringlist (output)` [W/L]
  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:

```
set $list$ = getOutStreamFromSection ('DosInAnIcon_netstart')

[DosInAnIcon_netstart]
net start
```

`$list1$` contains among some surrounding stuff the list of all mounted shares of a PC.
see also : [getReturnListFromSection]

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]
  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
```

see also : [shellCall_list]

- `shellCall (`<command string>`) : noresult` //since 4.11.6.1 [W/L]

```
shellCall('net start')
```

Is a shortcut for this expression:

```
DosInAnIcon_netstart winst /sysnative

[DosInAnIcon_netstart]
net start
```

see also : [shellCall]

- `shellCall (<command string>) : string (exitcode)` //since 4.11.6.1 [W/L]

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

Is a shortcut for this expression:

```
DosInAnIcon_netstart winst /sysnative
set $exitcode$ = getLastExitcode

[DosInAnIcon_netstart]
net start
```

see also : [shellCall_str]

- `getReturnListFromSection (`section name)`` [W/L]
  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

```
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 10.7 in this manual).
Or the list of opsi clients is produced by the reference to a opsi service call:

```
DefStringList $result$
Set $result$=getReturnListFromSection("opsiservicecall_clientIdsList")

[opsiservicecall_clientIdsList]
"method":"getClientIds_list"
"params":[]
```

see also : [getOutStreamFromSection]

9.5.6 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 OF 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:

```
Set  $list$ = GetRegistryVarMap32("hklm\software\opsi.org\opsi-script-test")
retrieving strings from GetRegistryVarMap32 [switch to loglevel 7 for debugging]
    (string   0)var1=value1
    (string   1)var2=value2
    (string   2)var3=value3
    (string   3)var4=444
    (string   4)var5=05 05 05 0F 10
    (string   5)var6=value6
```

Given the registry entries in the example above, and the following code:

```
set $list$ = getRegistryKeyList32("hklm\software\opsi.org\opsi-script-test")
```

we will get the following key in the log:

```
Set  $list$ = GetRegistryKeyList32("hklm\software\opsi.org\opsi-script-test")
Registry started with redirection (32 Bit)
    retrieving strings from GetRegistryKeyList32 [switch to loglevel 7 for debugging]
        (string   1)key1
        (string   2)key2
        (string   3)key3
        (string   4)key4
```

9.5.7 Stringlists from the Product Properties [W/L]

- `getProductPropertyList`(<propname>,<default value>)
  returns a stringlist of values that are referred to by the multivalue product property <propname>. If there is no connection to the opsi server, then the resulting stringlist contains only <default value>.
  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.
  `<default value>` decribes 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. Available since 4.11.3
  Example:

```
;Property "dummymulti" has the values: ("ab", "cd", "ef", "g,h")
set $list$ = GetProductPropertyList ("dummymulti","True")
if not ("" = takeFirstStringContaining($list$,"g,h"))
        comment "GetProductPropertyList passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "GetProductPropertyList failed"
endif

set $ConstTest$ = "ab,cd,ef,g,h"
set $CompValue$ = GetProductProperty ("dummymulti","True")
if ($ConstTest$ = $CompValue$)
        comment "GetProductProperty passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "GetProductProperty failed"
endif

;;;;;;another Example to get a list as default-property

DefStringList $list$
DefStringList $propertyList$
Set $propertyList$ = createStringList('ab','cd','de')
Set $list$ = GetProductPropertyList ("dummymulti",$propertyList$)
```

9.5.8 Other String Lists [W/L]

- `getProfilesDirList : stringlist` //since 4.11.3.2 [W/L]
  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]
  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]: *PID*
  – [W]: *Domain/User.* [L]: *User*
  – [L]: *full command line of the process*

9.5.9 Transforming String Lists [W/L]

- `getSubList (`<start index> : <end index>, <list>`) : stringlist` [W/L]
  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(:, $list$)
```

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*.

- `getListContaining(`<list>,<search string>`) : stringlist` [W/L]
  returns the first string from <list> which contains <search string>. Returns empty string if <seach string> is not found.

- `getListContainingList(`<list1>,<list2>`) : stringlist` //since 4.11.3.7 [W/L]
  returns the intersection of list1 and list2.

- `getSubListByMatch (`<search string>, <target list>`)` :stringlist //since 4.12.0.14 [W/L]
  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]
  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]
  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]
  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]
  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]
  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]
  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=vakue, the complete string will be part of the result list.

- `takeFirstStringContaining`(<list>,<search string>) : `string` [W/L]
  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]
  Appends <string> to the list <list>.

- `addListToList`(<dest list>,<src list>) : `stringlist` //since 4.10.8 [W/L]
  Appends the list <list2> to the list <list1>.

- `reverse` (<list>) : `stringlist` [W/L]
  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]
clears the list.

- `reencodestrlist`(<list>, <from>, <to>) : `stringlist` //since 4.11.4.2 [W/L]
  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 Section 6.3.

- `removeFromListByContaining(`<search string>`, `<target list>`) :  stringlist` //since 4.11.5.1 [W/L]
  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]
  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:

```
comment ""
comment "-------------------------------"
comment "Testing: "
message "removeFromListByContaining"
set $string1$ = "%Scriptpath%\test-files\encoding\10lines.txt"
set $list1$ = loadTextFileWithEncoding($string1$, "cp1252")
comment "search with string"
comment "search with string constant"
set $ConstTest$ = "9"
set $list2$ = removeFromListByContaining("line 5", $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

set $ConstTest$ = "9"
set $list2$ = removeFromListByContaining("LINE 5", $list1$)
comment "the match is case insensitive"
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif


set $ConstTest$ = "0"
set $list2$ = removeFromListByContaining("line", $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

set $ConstTest$ = "8"
comment "searchstr 1 will found in 'line 1' and 'line 10'"
set $list2$ = removeFromListByContaining("1", $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

comment "search with string function"
set $ConstTest$ = "9"
set $list2$ = removeFromListByContaining(trim(" line 5 "), $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

comment "search with string variable"
set $string1$ = "line 5"
set $ConstTest$ = "9"
set $list2$ = removeFromListByContaining($string1$, $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

comment "search with string list"
comment "search with string list variable"
set $list3$ = createStringList ('1', '2', '3', '4', '5')
comment "searchstr 1 will found in 'line 1' and 'line 10'"
set $ConstTest$ = "4"
set $list2$ = removeFromListByContaining($list3$, $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

comment "search with string list variable"
comment "searchstr 1 will found in 'line 1' and 'line 10'"
set $ConstTest$ = "4"
set $list2$ = removeFromListByContaining(createStringList ('1', '2', '3', '4', '5'), $list1$)
set $CompValue$ = count($list2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `removeFromListByMatch`(<searchstring>,<target list>) : `stringlist` //since 4.11.6 [W/L]
  Returns a copy of <target list> where all lines are removed that exactly match a line out of <search list>. The match to <search sting> is case insensitiv.
  see also : [removeFromListByContaining_str]
  see also : [removeFromListByContaining_list]

- `setStringInListAtIndex`(<newstring>,<list>,<indexstr>) : `stringlist` //since 4.11.6 [W/L]
  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]

9.5.10 Iterating through String Lists [W/L]

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.

Caution
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.

9.6 opsiservicecall and json Related functions [W/L]

- `jsonIsValid(`<jsonstr>`) : boolean` //since 4.11.6: [W/L]
- `jsonIsArray(`<jsonstr>`) : boolean` //since 4.11.6: [W/L]
- `jsonIsObject(`<jsonstr>`) : boolean` //since 4.11.6: [W/L]
- `jsonAsObjectHasKey(`<jsonstr>,<keystr>`) : boolean` //since 4.11.6: [W/L]
- `jsonAsArrayCountElements(`<jsonstr>`) : intstr` //since 4.11.6: [W/L]
- `jsonAsObjectCountElements(`<jsonstr>`) : intstr` //since 4.11.6: [W/L]
- `jsonAsArrayGetElementByIndex(`<jsonstr>, <indexstr>`) : jsonstring` //since 4.11.6: [W/L]
- `jsonAsObjectGetValueByKey(`<jsonstr>, <keystr>`) : valuestring` //since 4.11.6: [W/L]
- `jsonAsObjectSetValueByKey(`<jsonstr>, <keystr>,<valuestring>`) : jsonstring` //since 4.11.6: [W/L]
- `jsonAsObjectSetStringtypeValueByKey(`<jsonstr>, <keystr>,<valuestring>`) : jsonstring` //since 4.11.6: [W/L]
- `jsonAsObjectDeleteByKey(`<jsonstr>, <keystr>`) : jsonstring` //since 4.11.6.4: [W/L]
- `jsonAsArrayPutObjectByIndex(`<jsonstr>, <indexstr>, <objectstr>`) : jsonstring` //since 4.11.6: [W/L]
- `jsonAsArrayDeleteObjectByIndex(`<jsonstr>, <indexstr>`) : jsonstring` //since 4.11.6.4: [W/L]
- `jsonAsArrayToStringList(`<jsonstr>`) : stringlist` //since 4.11.6: [W/L]
- `jsonStringListToJsonArray(`<strlist>`) : jsonstr` //since 4.11.6: [W/L]
- `jsonAsObjectGetKeyList(`<jsonstr>`) : stringlist` //since 4.11.6: [W/L]

Example: Restoring productOnClient entries from a file to the server:

```
DefVar $poc_file$
DefVar $objectStr$
DefVar $ArrayStr$
DefVar $pid$

DefStringlist $resultlist$
DefStringlist $resultlist1$
DefStringlist $productIdList$
DefStringlist $pocList$


Message "Delete productOnClient from opsi backend ..."
set $resultlist$ = getReturnListFromSection("opsiservicecall_getPOC")
Set $ArrayStr$ = takestring(0, $resultlist$)
if not(jsonIsValid($ArrayStr$))
        LogError "got no valid json from Service"
        isFatalError
endif
if not(jsonIsArray($ArrayStr$))
        LogError "got no json Array from Service"
        isFatalError
endif
comment "extract productIds ..."
comment "clean target list"
set $productIdList$ = emptylist($productIdList$)
comment "get stringlist "
set $pocList$ = jsonAsArrayToStringList($ArrayStr$)
for %aktpoc% in $pocList$ do sub_fill_product_ids
for %aktProductId% in $productIdList$ do opsiServiceCall_del_productOnClient

Message "Restore productOnClient from file ..."
comment " get Restore data from file ..."
Set $ArrayStr$ = strLoadTextFile($poc_file$)
if not(jsonIsValid($ArrayStr$))
        LogError "got no valid json from file"
        isFatalError
endif
if not(jsonIsArray($ArrayStr$))
        LogError "got no json Array from file"
        isFatalError
endif

comment "get list from array"
set $pocList$ = jsonAsArrayToStringList($ArrayStr$)
comment "loop over list"
for %pocindex% = "0" to calculate(count($pocList$)+"-1") do sub_set_clientid_in_poclist
comment "convert modified list to jason array"
set $ArrayStr$ = jsonStringListToJsonArray($pocList$)
set $ArrayStr$ = unquote2($ArrayStr$,"[]")
comment "write back"
opsiServiceCall_updatePOC

[sub_fill_product_ids]
set $objectstr$ = '%aktpoc%'
set $pid$ = jsonAsObjectGetValueByKey($objectstr$, "productId" )
set $productIdList$ = addToList($productIdList$,$pid$)

[sub_set_clientid_in_poclist]
set $objectStr$ = takeString("%pocindex%", $poclist$)
set $objectStr$ = jsonAsObjectSetStringtypeValueByKey(($objectStr$, "clientId","%opsiserviceUser%
    ")
set $poclist$ = setStringInListAtIndex($objectStr$,$poclist$,"%pocindex%")

[opsiServiceCall_updatePOC]
"method": "productOnClient_updateObjects"
"params": [
                                        '$ArrayStr$',
                                        ]

[opsiservicecall_getPOC]
        "method": "productOnClient_getObjects"
        "params":[
           "[]",
           '{"clientId":"%opsiserviceUser%","productType":"LocalbootProduct"}'
           ]

[opsiServiceCall_del_productOnClient]
"method": "productOnClient_delete"
"params": [
                                        '%aktProductId%',
                                        '%opsiserviceuser%',
                                        ]
```

9.7 Calculating with numbers [W/L]

opsi-winst scripts do not have a special type of varibles for numbers. But there are some functions to help calculating with numbers.

- `calculate`(<arithmetic string expression>) `:  string (number)`
  this string function calculates the arithemtic expression of the string <str> and returns the rounded result as a string.
  Internally the calculations are done with real numbers. This function acceptsthe 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 funktion `isNumber`.
  (since version 4.11.3.5)

Example:

```
set $ConstTest$ = "0"
set $CompValue$ = calculate("-1+1")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

```
set $ConstTest$ = "1"
set $CompValue$ = calculate("0+1")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $ConstTest$ = "-1"
set $CompValue$ = calculate("0-1")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "5"
set $ConstTest$ = "25"
set $CompValue$ = calculate($string1$+"*"+$string2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "5"
set $ConstTest$ = "1"
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "0"
set $ConstTest$ = ""
comment " expecting devision by zero error and empty string result"
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "9"
set $string2$ = "10"
set $ConstTest$ = "1"
comment "result 0.9 is rounded to 1 "
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
```

```
endif
set $string1$ = "10"
set $string2$ = "9"
set $ConstTest$ = "1"
comment "result 1.1111 is rounded to 1 "
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "5"
set $ConstTest$ = "55"
comment " rule * before +"
set $CompValue$ = calculate($string1$+"+"+$string2$+"*10")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "5"
set $ConstTest$ = "100"
comment "brackets before  rule * before + "
set $CompValue$ = calculate("("+$string1$+"+"+$string2$+")*10")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "ten"
set $ConstTest$ = ""
comment "invalid char error"
set $CompValue$ = calculate($string1$+"*"+$string2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = ""
set $ConstTest$ = "25"
comment "5* is interpreted as 5*5"
set $CompValue$ = calculate($string1$+"*")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = ""
set $ConstTest$ = "10"
comment "5+ is interpreted as 5+5"
set $CompValue$ = calculate($string1$+"+")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "nothing"
set $string2$ = "foo"
set $ConstTest$ = ""
comment "invalid char error"
set $CompValue$ = calculate($string1$+"*"+$string2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $string1$ = "5"
set $string2$ = "foo"
set $ConstTest$ = ""
comment "invalid char error"
set $CompValue$ = calculate($string1$+"/"+$string2$)
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

For more examples refer to the product *opsi-script-test* at the section *$Flag_calculate$ = "on"*

9.8 Process and Script Related functions [W/L]

- `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 ist **not recommended**.

- `processIsRunning(`<process name>`) : boolean` //since 4.11.6.1 [W/L]
  Returns `true` if <process name> is found in the process list

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

```
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]

```
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]
  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 `Retricted` which do not allow to run any unsigned scripts. In order to run scripts the `powershellCall` function does by default the following: The current execution-policy is backuped 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>.

Example:

```
set $list$= powershellCall_list('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*]]) : `noresult` //since 4.12.0.16 [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*]]) : `string (exitcode)` //since 4.12.0.16 [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]
  Starts the command binary <string> as process and returns te exitcode

```
set $exitcode$ = processCall('setup.exe /S')
```

Is a shortcut for this expression:

```
Winbatch_setup
set $exitcode$ = getLastExitcode

[Winbatch_setup]
setup.exe /S
```

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-winst/opsi-script* shall proceed while the started processes run in their own threads.

- `/TimeOutSeconds` <seconds>
  A timeout setting. After waiting <seconds>, *opsi-winst/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-winst/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-winst/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 pathes.
  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 interpretiert.
  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*.

9.9 Special Commands [W/L]

- `Killtask` <process name> ` : noresult` [W/L]
  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]
  Set the given directory as working directory of the *opsi-winst/opsi-script*. Affects all subsequent actions (e.g. winbatch sections) and will be reset at the end of a script. Beispiel :

```
ChangeDirectory "%SCRIPTPATH%\programm"
```

- `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 rocess that may have changed the environment (e.g. Registry section or setup program). Even if the program that runs after the opsi-script get 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:

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

- `markTime` [W/L]
  sets a time stamp for the current system time and logs it.

- `diffTime` [W/L]
  logs the time passed since the last `marktime`.

9.9.1 Commands to control the logging

- `comment` <string> or `comment` = <const string> [W/L]
  writes the value of the String expression resp. the sequence of characters into the log file.

- `LogError` <string> or `LogError` = <const string> [W/L]
  writes additional error messages to the log file and increments the error counter by one.

- `LogWarning` <string> or `LogWarning` = <const string> [W/L]
  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]

- `includelog` <file name> <tail size> [<encoding>] //since 4.11.4.1 [W/L]
  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-winst 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: Encoding

- `SetConfidential` <secret string> [W/L]
  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]
  This function should be used to get confidentail 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 inital 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: This is a ***(confidential)*** string
comment: should be in the log file: This is a ***(confidential)*** string----
```

see also : [SetConfidential] see also : [GetConfidentialProductProperty]

9.10 Commands for User Information and User Interaction [W/L]

- `Message` <string expression>
  bzw.
  `Message` = <sequence of characters>
  lets *opsi-winst/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.

- `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]
  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 ActionProgreas is displayed at the configed.

9.11 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-winst/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 excuted for the actual user. The inrormation 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 booleasn function returns *true* if the script is running as *userLoginScript*. See also: `GetScriptMode`
  see also : [GetScriptMode]

9.12 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]

The temporary varibale %<temporary string variable>% must not be declared and is available in the called sub-section as constant.

Example:

Code from opsi-script-test:

```
message "for to loop"
set $ConstTest$ = "12345"
set $CompValue$ = ""
for %s% = "1" to "5" do sub_iteration_test
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif

[sub_iteration_test]
set $CompValue$ = $CompValue$ + '%s%'
```

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: ""

~~~~~~ Looping through:  '1', '2', '3', '4', '5'

  ~~~~~~~ Start Sub ~~~~~~~  sub_iteration_test
  Set  $CompValue$ = $CompValue$ + '1'
    The value of the variable "$CompValue$" is now: "1"

  ~~~~~~~ End Sub   ~~~~~~~  sub_iteration_test


  ~~~~~~~ Start Sub ~~~~~~~  sub_iteration_test
  Set  $CompValue$ = $CompValue$ + '2'
    The value of the variable "$CompValue$" is now: "12"

  ~~~~~~~ End Sub   ~~~~~~~  sub_iteration_test


  ~~~~~~~ Start Sub ~~~~~~~  sub_iteration_test
  Set  $CompValue$ = $CompValue$ + '3'
    The value of the variable "$CompValue$" is now: "123"

  ~~~~~~~ End Sub   ~~~~~~~  sub_iteration_test
```

```
  ~~~~~~~ Start Sub ~~~~~~~  sub_iteration_test
  Set  $CompValue$ = $CompValue$ + '4'
    The value of the variable "$CompValue$" is now: "1234"

  ~~~~~~~ End Sub   ~~~~~~~  sub_iteration_test


  ~~~~~~~ Start Sub ~~~~~~~  sub_iteration_test
  Set  $CompValue$ = $CompValue$ + '5'
    The value of the variable "$CompValue$" is now: "12345"

  ~~~~~~~ End Sub   ~~~~~~~  sub_iteration_test


~~~~~~ End Loop
If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: passed
Else
EndIf
```

9.13 Switch / Case Statement [W/L]

Syntax:

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

Examples:

Code from opsi-script-test:

```
set $ConstTest$ = "5"
Switch $ConstTest$
        Case "1"
                set $CompValue$ = "1"
        EndCase
        Case "2"
                set $CompValue$ = "2"
        EndCase
        Case "3"
                set $CompValue$ = "3"
        EndCase
        Case "4"
                set $CompValue$ = "4"
        EndCase
        Case "5"
```

```
                set $CompValue$ = "5"
        EndCase
        Case "6"
                set $CompValue$ = "6"
        EndCase
        Case "7"
                set $CompValue$ = "7"
        EndCase
        DefaultCase
                set $CompValue$ = "notexisting"
        EndCase
EndSwitch
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

.

```
[Sub_check_exitcode]
comment "Test for installation success via exit code"
set $ExitCode$ = getLastExitCode
; informations to exit codes see
; http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx
; http://msdn.microsoft.com/en-us/library/aa368542.aspx
Switch $ExitCode$
        Case "0"
                comment "Looks good: setup program gives exitcode zero"
        EndCase
        Case "1605"
                comment "ERROR_UNKNOWN_PRODUCT  1605"
                comment "This action is only valid for products that are currently installed."
                comment "Uninstall of a not installed product failed - no problem"
        EndCase
        Case "1641"
                comment "looks good: setup program gives exitcode 1641"
                comment "ERROR_SUCCESS_REBOOT_INITIATED 164"
                comment "The installer has initiated a restart."
                comment "This message is indicative of a success."
                ExitWindows /Reboot
        EndCase
        Case "3010"
                comment "looks good: setup program gives exitcode 3010"
                comment "ERROR_SUCCESS_REBOOT_REQUIRED  3010"
                comment "A restart is required to complete the install."
                comment "This message is indicative of a success."
                ExitWindows /Reboot
        EndCase
        DefaultCase
                logError "Fatal: Setup program gives an unknown exitcode unequal zero: " + $
    ExitCode$
                isFatalError "Exit Code: "+ $ExitCode$
        EndCase
EndSwitch
```

9.14 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

```
;Which Windows version?
DefVar $MSVersion$

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

9.14.1 General Syntax

The syntax of the complete `if` statement is:
`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-winst/opsi-script* script).

9.14.2 Boolean Expressions

The String comparison (which is a Boolean expression) has the form
`<String expression> <comparison sign> <String expression>`
where <comparison sign> is one of the signs
`< <= = >= >`

String comparisons in *opsi-winst/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:
`<String expression> INT<comparison sign> <String expression>`
Such, we can build expressions as

```
if $Name1$ INT<= $Name2$
```

or

```
if $Number1$ INT>= $Number2$
```

Boolean operators are `AND`, `OR`, and `NOT()` (case does not matter). If b1, b2 and b3 are Boolean expressions the combined expressions
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]
  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

- `LineExistsIn (`<string>, <file name>`) : bool` [W/L]
  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]
  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]
  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-winst/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.

- `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.

- `HasMinimumSpace`(<Laufwerksname>, <Kapazität>)
  returns true if at least a capacity capacity is left on drive drivename. capacity as well as drivename 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]
  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]
  Boolean function which returns *true* if <str> represents an integer.

- `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]
  true if the running OS is a Windows PE

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

- `saveTextFile(`<list>, < filename>`) :  bool` [W/L]
  true: if list is succesfully written to file

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

- `CompareDotSeparatedNumbers(`<str1>,<relation str>,<str2>`) : bool` //since 4.11.5.2: [W/L]
  compares two strings of the form <number>[.<number>[.<number>[.<number>]]]
  by the <relation str> which may be one of [<,<=,=,>=,>].
  see also: string function `CompareDotSeparatedNumbers(`<string1>, <string2>`)` : see also : [CompareDotSeparatedNumbers_str]

Example:
The code:

```
set $string1$ = "1.2.30.4.5"
set $string2$ = "1.20.30.4.5"
if CompareDotSeparatedNumbers($string1$, "<", $string2$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
if CompareDotSeparatedNumbers($string1$, "<=", $string2$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
if CompareDotSeparatedNumbers($string1$, "=<", $string2$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
if CompareDotSeparatedNumbers($string1$, "=", $string2$)
        set $TestResult$ = "not o.k."
        LogWarning "failed"
else
        comment "passed"
endif
if CompareDotSeparatedNumbers($string1$, ">=", $string2$)
        set $TestResult$ = "not o.k."
        LogWarning "failed"
else
        comment "passed"
endif
if CompareDotSeparatedNumbers($string1$, "=>", $string2$)
        set $TestResult$ = "not o.k."
        LogWarning "failed"
else
        comment "passed"
endif
if CompareDotSeparatedNumbers($string1$, ">", $string2$)
        set $TestResult$ = "not o.k."
        LogWarning "failed"
else
        comment "passed"
endif
```

produce the log:

```
Set  $string1$ = "1.2.30.4.5"
  The value of the variable "$string1$" is now: "1.2.30.4.5"
Set  $string2$ = "1.20.30.4.5"
  The value of the variable "$string2$" is now: "1.20.30.4.5"
If
    Checking if "1.2.30.4.5" is "<" than / as "1.20.30.4.5"
  CompareDotSeparatedNumbers($string1$, "<", $string2$)   <<< result true
Then
  comment: passed
Else
EndIf
If
    Checking if "1.2.30.4.5" is "<=" than / as "1.20.30.4.5"
  CompareDotSeparatedNumbers($string1$, "<=", $string2$)   <<< result true
Then
  comment: passed
Else
EndIf
If
    Checking if "1.2.30.4.5" is "=<" than / as "1.20.30.4.5"
  CompareDotSeparatedNumbers($string1$, "=<", $string2$)   <<< result true
Then
  comment: passed
Else
EndIf
If
    Checking if "1.2.30.4.5" is "=" than / as "1.20.30.4.5"
  CompareDotSeparatedNumbers($string1$, "=", $string2$)   <<< result false
Then
Else
  comment: passed
EndIf
If
    Checking if "1.2.30.4.5" is ">=" than / as "1.20.30.4.5"
  CompareDotSeparatedNumbers($string1$, ">=", $string2$)   <<< result false
Then
Else
  comment: passed
EndIf
If
    Checking if "1.2.30.4.5" is "=>" than / as "1.20.30.4.5"
  CompareDotSeparatedNumbers($string1$, "=>", $string2$)   <<< result false
Then
Else
  comment: passed
EndIf
If
    Checking if "1.2.30.4.5" is ">" than / as "1.20.30.4.5"
  CompareDotSeparatedNumbers($string1$, ">", $string2$)   <<< result false
Then
Else
  comment: passed
EndIf
```

- `CompareDotSeparatedStrings(`<str1>,<relation str>,<str2>`) : bool` //since 4.11.5.2: [W/L]

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]`

- `boolToString(`<boolean expression>`)` : bool string (true/false) // since 4.12.0.0 [W/L]

- `stringToBool(`<string expression: true/false>`)` : boolean // since 4.12.0.0 [W/L]

- `RegKeyExists(`<regkey>[,<access str>]`)` : `bool` //since 4.12.0.16 [W]
  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> [,<access 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")
```

9.15 Include Commands

Caution
Using include commands can lead to confusing code.
Use with caution if you are a beginner.

9.15.1 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-winst/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]
- A existing file in `%ScriptPath%` [W/L]
- A file in `%opsiScriptHelperPath%\lib` [W]
  Is equivalent to: *%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib*
- A existing file in `%ScriptPath%/../lib` [W/L]
- A existing file in `%WinstDir%\lib` [W]

The tests for the location of the <file name> are done in the order above. *opsi-winst/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-winst 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"
```

9.15.2 Include Commands: Library

The following include files are shipped with version 4.11.3, and are located in `%WinstDir%\lib`:

`insert_check_exit_code.opsiinc`:

```
; opsi include file

DefVar $ExitCode$

include_append "section_sub_check_exitcode.opsiinc"
```

`insert_get_licensekey.opsiinc`:

```
; opsi include file

DefVar $LicenseRequired$
DefVar $LicenseKey$
DefVar $LicensePool$

include_append "section_sub_get_licensekey.opsiinc"
```

`section_sub_check_exit_code.opsiinc`:

```
;opsi include file

[Sub_check_exitcode]
comment "Test for installation success via exit code"
set $ExitCode$ = getLastExitCode
; informations to exit codes see
; http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx
; http://msdn.microsoft.com/en-us/library/aa368542.aspx
if ($ExitCode$ = "0")
        comment "Looks good: setup program gives exitcode zero"
else
        comment "Setup program gives a exitcode unequal zero: " + $ExitCode$
        if ($ExitCode$ = "1605")
                comment "ERROR_UNKNOWN_PRODUCT  1605    This action is only valid for products
    that are currently installed."
                comment "Uninstall of a not installed product failed - no problem"
        else
                if ($ExitCode$ = "1641")
                        comment "looks good: setup program gives exitcode 1641"
                        comment "ERROR_SUCCESS_REBOOT_INITIATED 1641    The installer has
    initiated a restart. This message is indicative of a success."
                        ExitWindows /Reboot
                else
                        if ($ExitCode$ = "3010")
                                comment "looks good: setup program gives exitcode 3010"
                                comment "ERROR_SUCCESS_REBOOT_REQUIRED  3010    A restart is
    required to complete the install. This message is indicative of a success."
                                ExitWindows /Reboot
                        else
                                logError "Fatal: Setup program gives an unknown exitcode unequal
    zero: " + $ExitCode$
                                isFatalError "Exit Code: "+ $ExitCode$
                        endif
                endif
        endif
endif
```

section_sub_get_licensekey.opsiinc:

```
; opsi include file

[Sub_get_licensekey]
if opsiLicenseManagementEnabled
        comment "License management is enabled and will be used"

        comment "Trying to get a license key"
        Set $LicenseKey$ = demandLicenseKey ($LicensePool$)
        ; If there is an assignment of exactly one license pool to the product the following call
     is possible:
        ; Set $LicenseKey$ = demandLicenseKey ("", $ProductId$)
        ;
        ; If there is an assignment of a license pool to a windows software id, it is possible to
     use:
        ; DefVar $WindowsSoftwareId$
        ; $WindowsSoftwareId$ = "..."
        ; Set $LicenseKey$ = demandLicenseKey ("", "", $WindowsSoftwareId$)

        DefVar $ServiceErrorClass$
        set $ServiceErrorClass$ = getLastServiceErrorClass
        comment "Error class: " + $ServiceErrorClass$

        if $ServiceErrorClass$ = "None"
                comment "Everything fine, we got the license key '" + $LicenseKey$ + "'"
        else
                if $ServiceErrorClass$ = "LicenseConfigurationError"
                        LogError "Fatal: license configuration must be corrected"
                        LogError getLastServiceErrorMessage
                        isFatalError $ServiceErrorClass$
                else
                        if $ServiceErrorClass$ = "LicenseMissingError"
                                LogError "Fatal: required license is not supplied"
                                isFatalError $ServiceErrorClass$
                        endif
                endif
        endif
else
```

```
        LogError "Fatal: license required, but license management not enabled"
        isFatalError "No Licensemanagement"
endif
```

9.16 Subprogram Calls

Statements in primary sections which refer to instructions declared elsewhere are subprogram calls.,

```
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:

9.16.1 Syntax of Procedure Calling

Formally, the syntax can be given by
<proc. type>(<proc. name> | <External proc. file> | <String list function>)

This expression may supplemented by one ore ore 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 sub program types is given at Section 9.16.

The second part determines where and how the lines of sub program are to be found.

1. The subprogram is a sequence of lines situated in the executed *opsi-winst/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-winst/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 searche rule is:
<file name> may be:

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

The tests for the location of the <file name> are done in the order above. *opsi-winst/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%/opsiorgkey.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 Chapter 10.

9.17 Controlling Reboot

The command `ExitWindows` is used to control reboots , shutdown and similar actions which should take place after the *opsi-winst/opsi-script* it self is terminated. The name of the command and the fact that there is no *ExitWindows* without modifier has histrical 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-winst/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-winst/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-winst/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-winst/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).

- `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 $WinstRegKey$
DefVar $RebootRegVar$

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
```

9.17.1 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 a *opsi-winst/opsi-script* script that the installation is failed we have to call the statement:
`isFatalError`
If this statement is called, then *opsi-winst/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:

```
DefVar $SpaceNeeded$"
Set $SpaceNeeded$" = "200 MB"

DefVar $LogErrorMessage$
Set $LogErrorMessage$ = "Not enough space on drive . Required "
Set $LogErrorMessage$ = $LogErrorMessage$ + $SpaceNeeded$"

if not(HasMinimumSpace ("%SYSTEMDRIVE%", $SpaceNeeded$))
  LogError $LogErrorMessage$
  isFatalError
  ; finish execution and set ProductState to failed

 else
```

```
  ; we start the installation
  ; ...
 endif
```

It is also possible to state
`isFatalError`
depending on the number of errors which occured 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 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
  `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:

```
markErrorNumber
; Errors occuring 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 occured"

else
    ; no error occured, lets log this:

    comment "no error occured"
endif
```

- `isSuccess` //since 4.11.3.7 [W/L]
  Abort the script as successful.

- `noUpdateScript` //since 4.11.3.7 [W/L]
  Do not run an update script after setup even if there is one.

- `isSuspended` //since 4.11.4.1 [W/L] Abort the script without notice to the server. The action request remain unchanged.

9.18 Local functions [W/L]

Since version 4.12, the opsi-script has also local functions.

An example:

```
DefFunc myFunc(val $str1$ : string, $str2$ : string) : string
        set $result$ = $str1$ + $str2$
endfunc
```

9.18.1 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.

9.18.2 Syntax

Definition

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

Where:

- `DefFunc` is the keyword used to start defining a local function..
- *<func name>* is the freely chosen name of the function.
- *calltype* is the call type of the parameter [`val` | `ref`]. `val`=*Call by Value*, `ref`=*Call by Reference*. Default: `val`
- *parameter* is the free selected name of the call parameter which is available as a local variable within the function under the aforementioned name.
- *ptype* is the type of data of the parameter 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.
  In this part there is the automaticly decared local variable `$result$` which should take the result of the function an so have the data type of the function.
- `endfunc` is the keyword used to end defining a local function.

A local function has to be defined **before** you can call the function.

9.18.3 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 `stringlist` which will deliver a `string` and a `stringlist`:

```
[actions]
DefVar $mystr$
DefVar $str1$
DefStringlist $list1$
DefStringlist $list2$

set $str1$ = 'ha'

DefFunc myFunc1(val $str1$ : string, $list1$ : stringlist) : stringlist
        set $result$ = createStringlist($str1$ , takeString(2,$list1$))
endfunc

set $list2$ = splitstring("/etc/opsi/huhu","/")
set $list1$ = myFunc1("hi",$list2$)
```

Expected results:

- $list1$ = [hi,opsi]

Function of type `string` to which a boolean `string` will be deliver:

```
[actions]

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:

```
[actions]
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:

```
[actions]
DefVar $mystr$

DefFunc myFunc4($str1$ : string) : string
        DefVar $locstr1$

        DefFunc myFunc5($str1$ : string) : string
                set $result$ = 'inner' + $str1$
        endfunc

        set $locstr1$ = '123'
        set $result$ = $str1$ + myFunc5($locstr1$)
endfunc

set $mystr$ = myFunc4("outer")
```

Expected results:

- *outerinner123*

Simple function of type `string` which pass a\` string\` by reference with a local variable:

```
[actions]
DefVar $mystr$
DefVar $str1$
DefVar $str2$

set $str1$ = 'ha'
set $str2$ = 'hi'

DefFunc myFunc6(ref $str1$ : string) : string
        DefVar $locstr1$
        set $locstr1$ = '123'
        set $str1$ = 'setinlocal'
        set $result$ = $locstr1$ + $str1$
endfunc

set $mystr$ = myFunc6($str2$)
set $mystr$ = $str1$ + $str2$
```

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:

```
[actions]
DefVar $mystr$
```

```
DefStringlist $list1$
DefStringlist $list2$

et $list2$ = splitstring("/etc/opsi/huhu","/")

DefFunc myFunc7(ref $list1$ : stringlist) : stringlist
        DefStringlist $loclist1$
        set $loclist1$ = splitstring("/a/b/c","/")
        set $list1$ = createStringList('setinlocal')
        set $loclist1$ = addListToList($loclist1$,$list1$)
        set $result$ = $loclist1$
endfunc

set $list1$ = myFunc7($list2$)
comment "$list2$ index 0: " + takestring(0,$list2$)
```

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:

```
[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:

```
[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:

```
[actions]
ScriptErrorMessages = false
DefVar $str1$

DefFunc myNoParamFunc() : string
        set $result$ = "huhu"
endfunc

set $str1$ = myNoParamFunc()
```

Expected results:

- $str1$ is: huhu

9.19 Import von Library Funktionen [W/L]

`importLib` <string expr> ; import library // since 4.12.0.0
<string expr> : <file name>[.<file extension>][`::`<function name>]
If no *.<file extension>* is given `.opsiscript` is used as default.
If no *::<function name>* is given, all function from the given file will be imported.

<file name> is:

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

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

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 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-winst/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.

10.1.1 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.

10.1.2 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

- `/AllNTUserProfiles` resp.

- `/AllNTUserSendTo`

Both variants mean:
The called Files section is executed once for each local Windows NT 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-winst/opsi-script* version 4.11.2 `%CurrentProfileDir%`. With option `/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 `/AllNTUserProfiles` if it is not running as *userLoginScript* (in *Machine* script mode) and without `/AllNTUserProfiles` 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

10.1.3 Commands

In a Files section the following commands are defined:

- `Copy` [W/L]
- `Delete` / `Del` [W/L]
- `SourcePath`
- `CheckTargetPath` [W/L]

- `chmod` [L]
- `hardlink` [W/L]
- `symlink` [W/L]
- `rename` [W/L]
- `move` [W/L]

`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.

  Caution
  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]
  - `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 *opsi-winst/opsi-script* default behaviour is therefore that a file in use will be marked for overwriting after the next reboot, AND the *opsi-winst/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.

  - `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 attributs).
  - `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` [-sfd[n]] <path>

or

- `Delete` [-sfd[n]] <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.

  Caution
  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 *opsi*. 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 *opsi* is a directory.
  It is safer to use the command `del` instead.

- `f` → force
  forces to delete read only files

- `c` → continue
  If a system file is in use, then it can be deleted only after a reboot. The *opsi-winst/opsi-script* default behaviour is therefore that a file in use will be marked for delete after the next reboot, AND the *opsi-winst/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.

- `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-exits`

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` <mask> <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]
  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]
  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]
  `move` <old filename> <new filename> // since 4.11.5 [W/L]
  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.
  At the moment it is not 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-winst/opsi-script* default behaviour is therefore that a file in use will be marked for overwriting after the next reboot, AND the *opsi-winst/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"
move "$HomeTestFiles$\files\temp\dummy2.txt" "$HomeTestFiles$\files\temp\move.txt"
```

10.2 Patches-Sections [W/L]

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]*.

10.2.1 Example

```
Patches_DUMMY.INI $HomeTestFiles$+"\dummy.ini"

[Patches_dummy.ini]
add [secdummy] dummy1=add1
add [secdummy] dummy2=add2
add [secdummy] dummy3=add3
add [secdummy] dummy4=add4
add [secdummy] dummy5=add5
add [secdummy] dummy6=add6
```

```
set [secdummy] dummy2=set1
addnew [secdummy] dummy1=addnew1
change [secdummy] dummy3=change1
del [secdummy] dummy4
Replace dummy6=add6 replace1=replace1
```

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
    addSection [secdummy]
      done
      done
  addEntry [secdummy] dummy2=add2
      done
  addEntry [secdummy] dummy3=add3
      done
  addEntry [secdummy] dummy4=add4
      done
  addEntry [secdummy] dummy5=add5
      done
  addEntry [secdummy] dummy6=add6
      done
  setEntry [secdummy] dummy2=set1
    Entry      dummy2=add2
    changed to dummy2=set1
  addNewEntry [secdummy] dummy1=addnew1
    appended entry
  changeEntry [secdummy] dummy3=change1
    entry      dummy3=add3
    changed to dummy3=change1
  delEntry [secdummy] dummy4
    in section secdummy deleted  dummy4=add4
  replaceEntrydummy6=add6 replace1=replace1
    replaced in line 7
  C:\tmp\testFiles\dummy.ini saved back
```

For more examples, please check the opsi standard product *opsi-script-test* and in this product the part *$Flag_winst_patches$ = "on"*

10.2.1.1 Call Parameter

The name of the file to be patched is passed as a parameter.

There are optional modifiers:

- `/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 `/AllNTUserProfiles` is implicit.
  In logscript mode, `%UserProfileDir%` will be interpreted as `%CurrentProfileDir%`.
  (Since Version 4.11.3.2)

10.2.2 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.

10.3 PatchHosts Sections [W/L]

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> <ipaddresse>
  sets the IP address for host <hostname> to <IPaddress>. If there is no entry for host name as yet it will be created.
- `setname` <ipaddresse> <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` <IPadresse> <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` <IPadresse> <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` <IPadresse>
  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>.

10.4 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.

10.5 PatchTextFile Sections [W/L]

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. Section 9.14.2).

10.5.1 Parameter

The text file which is to be treated is given as parameter.

There are optional modifiers:

- `/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 `/AllNTUserProfiles` is implied. In the *Loginscript* Mode the `%UserProfileDir%` is interpreted as `%CurrentProfileDir%`.
  (since version 4.11.3.5)

10.5.2 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>".
  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-winst/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.
- `setKeyValueSeparator` <separator char> //since 4.11.4.4
  sets for key/value pairs (command `setValueByKey`) the separator char (Default is =)
- `setValueByKey` <keystr> <valuestr> //since 4.11.4.4
  looks for a key/value pair with the key <keystr> and set here as value <valuestr>. Is <keystr> not found, the entry will be created at the cursor position.

10.5.3 Examples

For more examples, please check the opsi standard product *opsi-script-test* and in this product the part *$Flag_winst_patch_text_file$ = "on"*

10.6 LinkFolder Sections [W/L]

In a LinkFolder section start menus entries as well as desktop links are managed.

10.6.1 LinkFolder Sections in Windows

E.g. the following section creates a folder named "acrobat“ in the common start menu (shared by all users):

```
[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
`set_subfolder ""`

In the third step, we can start setting links. The command is a multi line expression starting with
`set_link`
and finished by `end_link`.

Between these lines the link parameters are defined in the following format:

`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 a 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 sensitiv) 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 there *Virtual Key Code* identifier. To get these identifier (as well as the allowed combinations) just use the following helper program:
  http://download.uib.de/opsi4.0/helper/showkeys.exe
  Keep in mind that a `shortcut` refernces the keys and not there contry specific layout. The Key `VK_OEM_3` is on an english keyboard the char *;* and on a german the letter *Ö.*
  Examples for allowed shurtcuts:
  - *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+O* (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*)

Caution
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

10.6.2 Examples

```
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:

```
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"*.

10.6.3 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:
`AudioVideo`, `Audio`, `Video`, `Development`, `Education`, `Game`, `Graphics`, `Network`, `Office`, `Settings`, `System`, `Utility`
The LinkFolder Sektion will work at Linux with different Desktop systems

10.7 XMLPatch Sections [W]

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-winst/opsi-script offers XMLPatch sections for editing XML documents.

With the actions defined for this section type *opsi-winst/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

10.7.1 Parameter

When calling an XMLPatch section the document path name is given as parameter, e.g.
`XMLPatch_mozilla_mimetypes $mozillaprofilepath$ + "\mimetypes.rdf"`

10.7.2 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 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:

```
<node_level_1_no_1>
    <node_level_2_A>
    </node_level_2_A>
    <node_level_2_B>
    </node_level_2_B>
    <node_level_2_C>
    </node_level_2_c>
</node_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

```
<node_level_1_no_2>hello world
</node_level_1_no_2>
```

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"
</node_level_1_no_1>
```

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 *opsi-winst/opsi-script*, selection based on criteria (1) to (3) and (7) is implemented

10.7.3 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 *opsi-winst/opsi-script* command is

- `OpenNodeSet`

There 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):

```
openNodeSet
  documentroot
  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:

```
openNodeSet 'define /handler value="doc"/application /'
```

In this syntax, the slash separates the steps into to 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:

```
openNodeSet

  documentroot
  all_childelements_with:
  all_childelements_with:
    elementname:"description"
    attribute:"type" value="browser"
    attribute:"name" value="mozilla"
  all_childelements_with:
    elementname:"linkurl"
    text:"http://www.mozilla.org"
end
```

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:

```
  - 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.

10.7.4 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.

10.7.5 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

```
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.

10.7.6 Examples

For further examples see the product *opsi-script-test* expecially the sector with *$Flag_winst_xml$ = "on"*

10.8 ProgmanGroups Sections

This section type is deprecated.

10.9 WinBatch-Sections [W/L]

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 desingned 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 depricated warning. Example:
ok: `notepad.exe test.txt`
depricated (not ok): `test.txt`

10.9.1 Call Parameter (Modifier)

There a several parameters of the WinBatch call which determine if (or how long) *opsi-winst/opsi-script* shall be wait for the started programs returning.

- `/WaitOnClose`
  Is the default
  opsi-winst/opsi-script waits for every initiated process to come back.
- `/LetThemGo`
  This is the contrary to `/WaitOnClose`. It is used if *opsi-winst/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-winst/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-winst/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-winst/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 pathes.
  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 interpretiert.
  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:

```
Winbatch_add_reg /64Bit
[Winbatch_add_reg]
"c:\windows\system32\regedit.exe" /s "%scriptpath%\my64.reg"
```

- `/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.

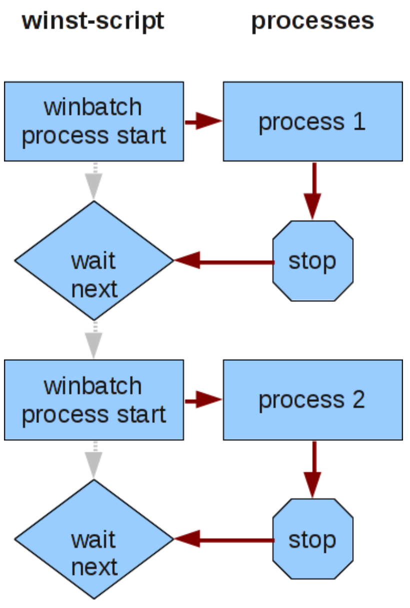


Figure 10.1: Sequential processing of a script that waits for the end of a program

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-winst/opsi-script*, the process is ended and the next command could be started.

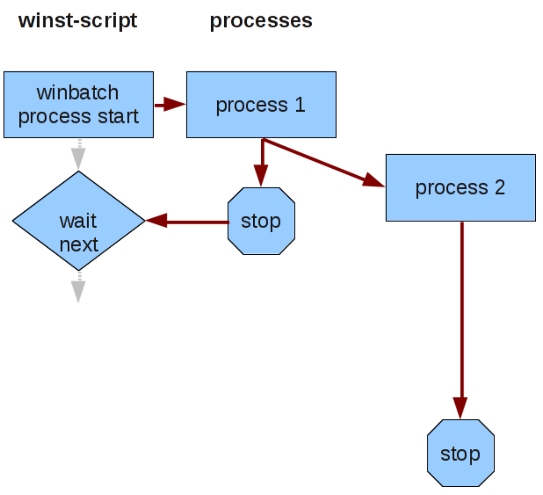


Figure 10.2: End of process while child process is still running

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.

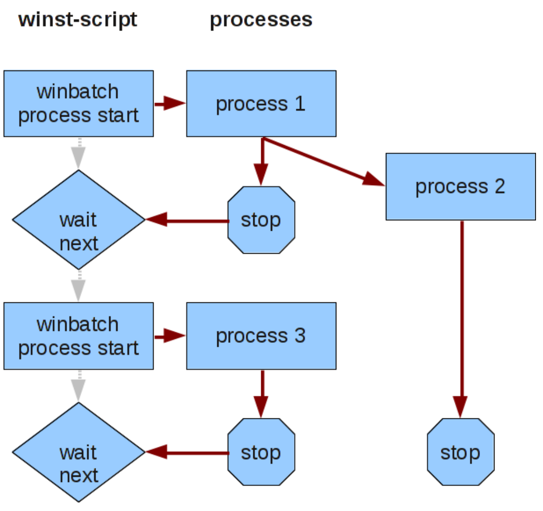


Figure 10.3: Overlapping of a child process and a parent process

Using the modifier `/WaitForProcessEnding` helps to avoid such a situation.

- `/TimeOutSeconds` <seconds>
  A timeout setting. After waiting <seconds>, *opsi-winst/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 *opsi-winst/opsi-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 *opsi-winst/opsi-script* waits until any process lets pop up a window with title window title. With the second option *opsi-winst/opsi-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 programms

10.9.2 Examples

For further examples see the product *opsi-script-test* expecially the sector with *$Flag_winst_winbatch$ = "on"*

10.10 DOSBatch/DosInAnIcon (ShellBatch/ShellInAnIcon) Sections [W/L]

Via DOSBatch (also called ShellBatch) sections a *opsi-winst/opsi-script* script uses Windows shell scripts for tasks which cannot be fulfilled by internal commands or for which already a batch script solution exists.

opsi-winst/opsi-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 `_winst<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:

- *opsi-winst/opsi-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.

Caution
Don't use commands that wait for user interaction.

10.10.1 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-winst/opsi-script* will handle the section.

The calling syntax is:

`Sektionsname [batch parameter] [winst [modifier]]`

Possible winst modifier are (since 4.11.1):

- `/32bit`
- `/64bit`
- `/Sysnative`
- `/showoutput` // since 4.11.4.6

These winst 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:

```
[Actions]
DosBatch_1 today we say "Hello World"

[DosBatch_1]
@echo off
echo %1 %2 %3 %4
pause
```

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*:

```
[Actions]
DosBatch_1 today we say winst /64bit

[DosBatch_1]
@echo off
echo %1 %2 %3 %4
pause
```

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:

```
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:

```
comment: Testing parameters for ShellBatch
Set  $ConstTest$ = "Hello world"
  The value of the variable "$ConstTest$" is now: "Hello world"
Set  $list$ = getOutStreamFromSection('DosInAnIcon_with_parameter world')

  DosInAnIcon_with_parameter
    c:\opsi.org\tmp\_opsiscript_Kj23Ej02.cmd saved back
    Executing "cmd.exe" /C c:\opsi.org\tmp\_opsiscript_Kj23Ej02.cmd world
    ExitCode 0

                output:
                ------------

                C:\Windows\system32>echo Hello world
                Hello world

    The file: c:\opsi.org\tmp\_opsiscript_Kj23Ej02.cmd has been deleted
    retrieving strings from getOutStreamFromSection [switch to loglevel 7 for debugging]
```

```
        (string   0)
        (string   1)C:\Windows\system32>echo Hello world
        (string   2)Hello world

Set  $CompValue$ = takeString(2,$list$)
    retrieving strings from $list$ [switch to loglevel 7 for debugging]
        (string   0)
        (string   1)C:\Windows\system32>echo Hello world
        (string   2)Hello world

  The value of the variable "$CompValue$" is now: "Hello world"
If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: ShellBatch parameter passed
Else
EndIf

comment: Testing parameters for ShellBatch
Set  $ConstTest$ = "Hello world"
  The value of the variable "$ConstTest$" is now: "Hello world"
Set  $tmp$ = "world"
  The value of the variable "$tmp$" is now: "world"
Set  $list$ = getOutStreamFromSection('DosInAnIcon_with_parameter $tmp$')

  DosInAnIcon_with_parameter
    c:\opsi.org\tmp\_opsiscript_Kz50Gi50.cmd saved back
    Executing "cmd.exe" /C c:\opsi.org\tmp\_opsiscript_Kz50Gi50.cmd world
    ExitCode 0

                output:
                ------------
>>>>>>> stable

                C:\Windows\system32>echo Hello world
                Hello world

    The file: c:\opsi.org\tmp\_opsiscript_Kz50Gi50.cmd has been deleted
    retrieving strings from getOutStreamFromSection [switch to loglevel 7 for debugging]
        (string   0)
        (string   1)C:\Windows\system32>echo Hello world
        (string   2)Hello world

Set  $CompValue$ = takeString(2,$list$)
    retrieving strings from $list$ [switch to loglevel 7 for debugging]
        (string   0)
        (string   1)C:\Windows\system32>echo Hello world
        (string   2)Hello world

  The value of the variable "$CompValue$" is now: "Hello world"
If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
  comment: ShellBatch parameter passed
Else
```

```
EndIf
```

10.10.2 Catch the output

The output of the shell commands can be captured by using the string list function `getOutStreamFromSection()` from the *opsi-winst/opsi-script*-scripts main-section see also:
Section 9.5.4).

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.

10.10.3 Examples

For further examples see the product *opsi-script-test* and there look at *$Flag_winst_dos$ = "on"*

10.11 Registry-Sections [W]

By a Registry section call we can create, patch and delete entries in the Windows registry. As usual, *opsi-winst/opsi-script* logs every operation in detail as long as logging is not turned off.

10.11.1 Examples

Let us set some registry variables by a call to the section Registry_TestPatch where the section is given by

```
[Registry_TestPatch]
openkey [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"*

10.11.2 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-winst/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 10.11.5).
- `/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 10.11.6).

These not *opsi-winst/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).

10.11.3 Commands

The default syntax of a Registry section is oriented at the command syntax of other patch operations in *opsi-winst/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-winst/opsi-script* syntax of the registry path name, the elements of a path are separated by single backslashs.

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-winst/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 `GetMultiSZFromFile` (cf. below).

Example for `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"
```

- `Add` <varname> = <value>

  resp.

  `Add` <varname> = <registry type> <value>
  are analogous to the `Set` commands with the difference that entries are only added but values of existing variables not changed.

- `Supp` <varname> <list separator> <supplement>
  This command interprets the string value of variable <varname>, a list of values separated by <list separator> and adds the string <supplement> to this list (if it not already contained). If <supplement> contains the <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).
  `Supp` keeps the original string variant (REG_EXPAND_SZ or REG_SZ).

Example:
The environment Path is determined by the value for the variable Path as defined inside the registry key

+ *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.

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

+ in order to supplement the path by *"C:\utils"* and *"%JAVABIN%"*.

+ (Windows expands %JAVABIN% to the correct path name if %JAVABIN% exists as variable and the String is a REG_EXPAND_SZ.)

+ Whom read the old value of Path from the environment variable, write this value to the registry value - and are then able to work with the registry variable:

+

```
[Actions]
DefVar $Path$
set $Path$ = EnvVar ("Path")
Registry_PathPatch

[Registry_PathPatch]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\control\Session Manager\Environment]
set "Path"="$Path$"
supp "Path"; "c:\orawin\bin"
```

+ CAUTION: The environment variable gets a changed value after a reboot or after a call of `UpdateEnvironment` see: [UpdateEnvironment]

- `GetMultiSZFromFile` <varname> <filename>
  reads the lines of a file and puts them together building a Multistring.
- `SaveValueToFile` <varname> <filename>
  exports the referred (String or MultiSZ) value as file <filename> lines (each String forming a line).
- `DeleteVar` <Varname>
  removes the entry with variable <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:

  ```
  [Registry_Keydel]
  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).

10.11.4 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 pathes 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:

```
[Registry_AllUsers]
openkey [Software\ORL\WinVNC3]
set "FileTransferEnabled"=reg_dword:0x00000000
```

Since *opsi-winst/opsi-script* version 4.11.2 you may give the root key *HKEY_CURRENT_USER* at the `openkey` command.
Example:

```
[Registry_AllUsers]
openkey [HKEY_CURRENT_USER\Software\ORL\WinVNC3]
set "FileTransferEnabled"=reg_dword:0x00000000
```

This has the folloing advantages:

- The syntax is easier to understand.
- The same registry section may be used with */AllNtuserdats* and in a *userLoginScript*

10.11.5 Registry Sections in Regedit Format

If a Registry section is called with parameter `/regedit` the section is not expected in *opsi-winst/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:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org]

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general]
"bootmode"="BKSTD"
"windomain"=""
"opsiconf"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\shareinfo]
"user"="pcpatch"
"pcpatchpass"=""
"depoturl"="\\\\bonifax\\opt_pcbin\\install"
"configurl"="\\\\bonifax\\opt_pcbin\\pcpatch"
"utilsurl"="\\\\bonifax\\opt_pcbin\\utils"
"utilsdrive"="p:"
"configdrive"="p:"
"depotdrive"="p:"
```

The sections denote registry keys to be opened. Each line describes some variable setting like the set command in a *opsi-winst/opsi-script* registry section.

But, we cannot really have an internal *opsi-winst/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-winst/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 converted. This can be done manually, e.g. by a suitable editor. But we may also feed the original file to *opsi-winst/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-winst/opsi-script* registry section. That is, a path like *[HKEY_CURRENT_USER\Software\ORL]* is to replaced by the relative *[Software\ORL]*.

10.11.6 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","",0,"AcroExch.FDFDoc"
HKCR,".pdf","",0,"AcroExch.Document"HKCR,"PDF.PdfCtrl.1","",0,"Acr"
```

10.12 OpsiServiceCall Sections [W/L]

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.

10.12.1 Call Parameters

Caution
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-winst/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 localhosts 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.

10.12.2 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

```
"method":<method name>
"params":[
        <params>
        ]
```

params is a (possibly empty) list of strings (comma-seperated). 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]
"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 9.5.5).

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 this manual: ####

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")
[opsiServiceCall_get_productOnClient_setup_objects]
"method": "productOnClient_getObjects"
"params": [
          "[]",
          '{"clientId":"%opsiserviceUser%","productType":"LocalbootProduct","actionRequest":"
    setup"}',
          ]
```

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.

```
DefVar $ArrayStr$
(...)
[opsiServiceCall_updatePOC]
"method": "productOnClient_updateObjects"
"params": [
          '$ArrayStr$'
          ]
```

10.12.3 Examples

For further examples watch the product *opsi-script-test* an there especially *$Flag_winst_opsiServiceCall$ = "on"*

10.13 ExecPython Sections [W/L]

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

```
execpython_hello -a "option a" -b "option b" "there we are"

[execpython_hello]
import sys
print "we are working in path: ", a
if len(sys.argv) > 1 :
        for arg in sys.argv[1:] :
                print arg
else:
  print "no arguments"


print "hello"
```

The print command output will be caught and written to the log file. So we get in the log

```
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.

10.13.1 Interweaving a Python Script with the opsi-winst Script

An execPython section is integrated with the surrounding *opsi-winst/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-winst/opsi-script* log.
- The *opsi-winst/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-winst/opsi-script* script.

An example for the first two ways of interweaving the python script with the *opsi-winst/opsi-script* script is already given above. We extend it to retrieve the values of some *opsi-winst/opsi-script* constants or variables.

```
[execpython_hello]
import sys
a = "%scriptpath%"
print "we are working in path: ", a
print "my host ID is ", "%hostID%"
if len(sys.argv) > 1 :
        for arg in sys.argv[1:] :
                print arg
else:
  print "no arguments"

print "the current loglevel is ", "$loglevel$"
print "hello"
```

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"')
```

10.13.2 Examples

For further examples watch the product *opsi-script-test* and there especially *$Flag_compare_to_python$ = "on"*

10.14 ExecWith Sections [W/L]

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.

10.14.1 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-winst/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*.

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-winst/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-winst/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`.

10.14.2 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:

```
ExecWith_close "%SCRIPTPATH%\autoit3.exe" WINST /letThemGo
```

A simple

```
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-winst/opsi-script* regards such lines as comments and eliminates them before handle).

The following example call the 64Bit version of the powershell.exe.

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

Note

For Powershell the script execution is disabled by default. So you have to enable it before you can use Execwith with powershell. In order to to that you shold call `powershell.exe set-executionpolicy RemoteSigned` in a DosInAnIcon.
Example

```
DosInAnIcon_setpolicy
ExecWith_powershell  powershell.exe
set $exitcode$ = getLastExitcode
if not ($exitcode$ = "0")
        comment "powershell script failed"
endif

[DosInAnIcon_setpolicy]
echo "powershell set-executionpolicy RemoteSigned ..."
powershell.exe set-executionpolicy RemoteSigned
exit %ERRORLEVEL%

[ExecWith_powershell]
echo "powershell opsi-script-test"
if ($?) {Exit(0)}
else {Exit(1)}
```

For further examples watch the product *opsi-script-test* and there especially *$Flag_autoit3_test$ = "on"*

10.15 LDAPsearch Sections [W]

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-winst/opsi-script* commands we do some explanations.

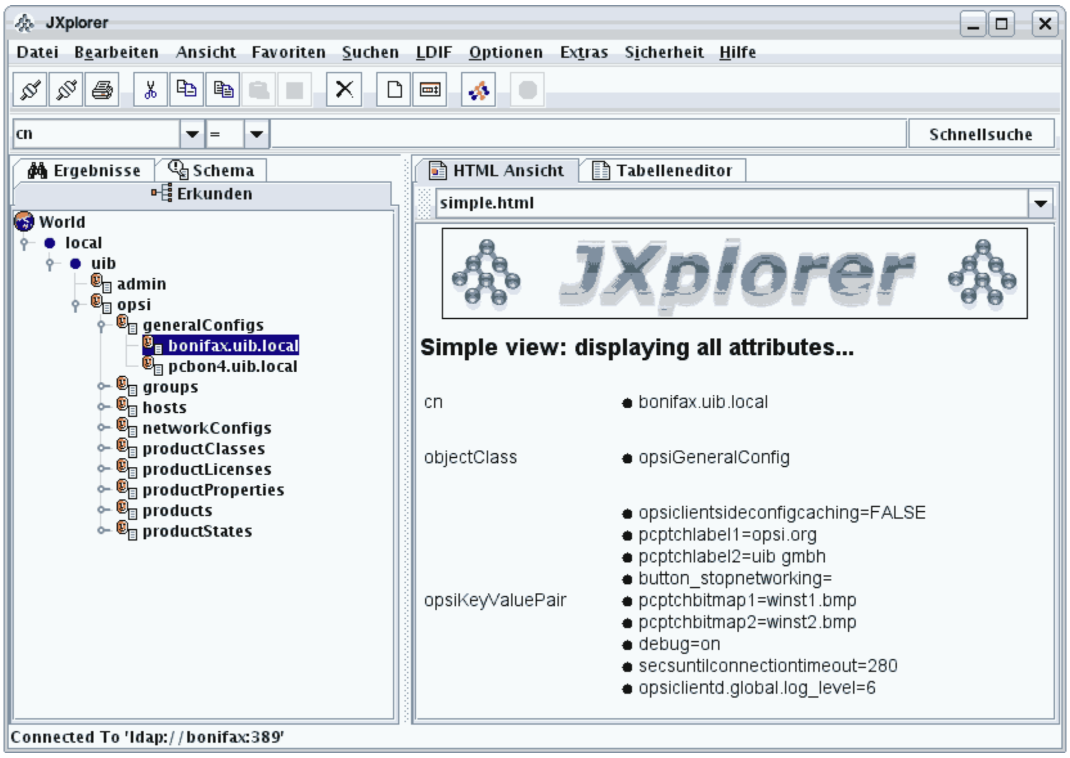
In subsection we give an example of the most probable usage of a LDAPsearch. The following subsections describe the syntax

10.15.1 LDAP – Protocol, Service, Directory

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):



Simple view: displaying all attributes...

cn	● bonifax.uib.local
objectClass	● opsiGeneralConfig
opsiKeyValuePair	● opsiclientsideconfigcaching=FALSE ● pcptchlabel1=opsi.org ● pcptchlabel2=uib gmbh ● button_stopnetworking= ● pcptchbitmap1=winst1.bmp ● pcptchbitmap2=winst2.bmp ● debug=on ● secsuntilconnectiontimeout=280 ● opsiclientd.global.log_level=6

Connected To 'ldap://bonifax:389'

Figure 10.4: 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 LDAPsearch roughly corresponding to the screenshot above as executed by *opsi-winst/opsi-script*.

Example

Using the *opsi-winst/opsi-script* section `ldapsearch_generalConfigs`:

```
[ldapsearch_generalConfigs]
targethost: bonifax
dn: cn=generalConfigs,cn=opsi,dc=uib,dc=local
```

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
        secsuntilconnectiontimeout=280
        opsiclientd.global.log_level=
```

There are several *opsi-winst/opsi-script* options to manage and reduce the complexity of the evaluation of such responses.

10.15.2 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`
- (no output option)

The output options determine the String list that is produced when a LDAPsearch section is called via getReturnlist-FromSection:

- 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 LDAPsearch 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.

10.15.3 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.

`ldapsearch_generalconfigs /cache`

executes the query and caches the response for further utilization.

Then, the call

`getReturnlistFromSection("ldapsearch_generalconfigs /cached /objects")`

produces the list

```
cn=generalconfigs,cn=opsi,dc=uib,dc=local
cn=pcbon4.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
cn=bonifax.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
```

If we narrow the tree selection by

```
[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.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
```

The corresponding attributes list contains three elements:

```
objectclass
cn
opsikeyvaluepair
```

In order to get the values associated to a single attribute we have to confine the query once more:

```
[ldapsearch_generalConfigs]
targethost: bonifax
dn: cn=bonifax.ubi.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
attribute: opsiKeyValuePair
```

The result now produced is an attributes list containing only one element. The corresponding values list looks like

```
opsiclientsideconfigcaching=false
pcptchlabel1=opsi.org
pcptchlabel2=uib gmbh
button_stopnetworking=
pcptchbitmap1=winst1.bmp
pcptchbitmap2=winst2.bmp
debug=on
secsuntilconnectiontimeout=280
opsiclientd.global.log_level=6
```

There are no LDAP means to reduce this result furthermore!

(But the *opsi-winst/opsi-script* function `getValue (key, list)` (cf. Section 9.5.4) may help in this case: E.g.
`getValue ("secsuntilconnectiontimeout", list)`
would produces 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".

10.15.4 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.
- `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-winst/opsi-script*. Default is "(objectclass=*)"
- `attributes:`
  A comma separated list of attribute names may be given. The default is to take any attribute.

10.15.5 Examples

A short and rather realistic example shall end this section:

$founditems$ be a StringList variable and *$opsiClient$* a String variable. The call of *getReturnlistFromSection* fetches the results of the section *ldapsearch_hosts*. The following code fragment returns the unique result for $opsiDescription$ if it exists. It reports an error if the search produces an unexpected result:

```
set $opsiClient$ = "test.uib.local"
set $founditems$ = getReturnlistFromSection("ldapsearch_hosts /values")

DefVar $opsiDescription$
set $opsiDescription$ = ""
if count(founditems) = "1"
  set $opsiDescription$ = takeString(0, founditems)
else
  if count(founditems) = "0"
    comment "No result found")
  else
    logError "No unique result for LdAPsearch for client " + $opsiclient$
  endif
endif


[ldapsearch_hosts]
targethost: opsiserver
targetport:
dn: cn=$opsiclient$,cn=hosts,cn=opsi,dc=uib,dc=local
typesOnly: false
filter: (objectclass=*)
attributes: opsiDescription
```

Example with user / password

```
comment ""
comment "------------------------------"
comment "Testing: "
comment "user / password"
Set $LdapHost$ = "vmix7.uib.local"
Set $LdapPort$ = "389"
Set $LdapUser$ = "cn=Administrator,cn=Users,dc=uib,dc=local"
Set $LdapPassword$ = "Linux123"
Set $LdapResultType$ = "objects"
Set $LdapSearchDn$ = "cn=Users,dc=uib,dc=local"
Set $LdapSearchAttributes$ = "name,objectClass"
Set $LdapFilter$ = "(&(objectclass=*))"

markErrorNumber
set $list1$ = getReturnListFromSection("ldapsearch_users /" + $LdapResultType$)
if errorsOccurredSinceMark > 0
        comment "failed while ldapsearch"
        set $TestResult$ = "not o.k."
else
        comment "passed"
endif

[ldapsearch_users]
targethost: $LdapHost$
```

```
targetport: $LdapPort$
user: $LdapUser$
password: $LdapPassword$
dn: $LdapSearchDn$
attributes: $LdapSearchAttributes$
filter: $LdapFilter$
```

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-winst/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-winst/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-winst/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 11.1: Constants

Constant	32 Bit	64 Bit
`%ProgramFilesDir%`	c:\program files	c:\program files (x86)
`%ProgramFiles32Dir%`	c:\program files	c:\program files (x86)
`%ProgramFiles64Dir%`	c:\program files	c:\program files
`%ProgramFilesSysnativeDir%`	c:\program files	c:\program files

`%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 aereas of registry and filesystem we have now the following new functions:

- `GetRegistrystringvalue32`
- `GetRegistrystringvalue64`
- `GetRegistrystringvalueSysNative`
- `FileExists32`
- `FileExists64`
- `FileExistsSysNative`

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-winst/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:

```
if $INST_SystemType$ = "64 Bit System"
        comment ""
        comment "-------------------------------"
        comment "Testing: "
        message "64 Bit redirection"
        Files_copy_test_to_system32
        if FileExists("%System%\dummy.txt")
                comment "passed"
        else
                LogWarning "failed"
                set $TestResult$ = "not o.k."
        endif
        ExecWith_remove_test_from_system32 'cmd.exe' /C
        Files_copy_test_to_system32 /64Bit
        if FileExists64("%System%\dummy.txt")
                comment "passed"
        else
                LogWarning "failed"
                set $TestResult$ = "not o.k."
        endif
        ExecWith_remove_test_from_system32 '%SystemRoot%\cmd64.exe' /C
endif
```

Registry Handling:

```
message "Write to 64 Bit Registry"
if ($INST_SystemType$ = "64 Bit System")
        set $ConstTest$ = ""
        set $regWriteValue$ = "64"
        set $CompValue$ = $regWriteValue$
        Registry_opsi_org_test /64Bit
        ExecWith_opsi_org_test "%systemroot%\cmd64.exe" /c
        set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByWinst")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
        set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByReg")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
        set $regWriteValue$ = "32"
        set $CompValue$ = $regWriteValue$
        Registry_opsi_org_test
        ExecWith_opsi_org_test "cmd.exe" /c
        set $ConstTest$ = GetRegistryStringValue("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByWinst")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
```

```
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
        set $ConstTest$ = GetRegistryStringValue("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByReg")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
else
        set $regWriteValue$ = "32"
        set $CompValue$ = $regWriteValue$
        Registry_opsi_org_test /64Bit
        ExecWith_opsi_org_test "cmd.exe" /c
        set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByWinst")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
        set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByReg")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
endif

if ($INST_SystemType$ = "64 Bit System")
        set $regWriteValue$ = "64"
        Registry_hkcu_opsi_org_test /AllNtUserDats /64Bit
        set $regWriteValue$ = "32"
        Registry_hkcu_opsi_org_test /AllNtUserDats
else
        set $regWriteValue$ = "32"
        Registry_hkcu_opsi_org_test /AllNtUserDats
        Registry_hkcu_opsi_org_test /AllNtUserDats /64Bit
endif
```

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 *opsi-winst/opsi-script* scripting.

12.1 9.1. Delete a File in all Subdirectories

Since *opsi-winst/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:

```
files_delete_Alt /allNtUserProfiles

[files_delete_Alt]
delete "%UserProfileDir%\alt.txt"
```

Neverthelesse we document a workaround which could be used in older *opsi-winst/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:

```
[Actions]

; variable for file name
DefVar $deleteFile$
set $deleteFile$ = "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

`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.

```
[Actions]
DefStringList $list0$
DefStringList $list1$
DefStringList $result$
Set $list0$=getOutStreamFromSection('DosBatch_netcall')
Set $list1$=getSublist(2:-3, $list0$)

DefVar $myservice$
DefVar $compareS$
DefVar $splitS$
DefVar $found$
Set $found$ ="false"
set $myservice$ = "opsiclientd"


comment "============================="
comment "search the list"
; for developping loglevel = 7
; setloglevel=7
; in normal use we dont want to log the looping
setloglevel = 5
for %s% in $list1$ do sub_find_myservice
setloglevel=7
comment "============================="

if $found$ = "false"
   set $result$ = getOutStreamFromSection ("dosinanicon_start_myservice")
endif


[sub_find_myservice]
set $splitS$ = takeString (1, splitStringOnWhiteSpace("%s%"))
Set $compareS$ = $splitS$ + takeString(1, splitString("%s%", $splitS$))
if $compareS$ = $myservice$
   set $found$ = "true"
```

```
endif


[dosinanicon_start_myservice]
net start "$myservice$"


[dosbatch_netcall]
@echo off
net start
```

12.3 Script for Installations in the Context of a Local Administrator

Sometimes it is necessary to run an installation script as an ordinary local user and not in the context of the opsi service. For example, there are installations that require a user context or use other services that are started after a user login.

MSI installations which seem to need a local user can sometimes be configured by the option *ALLUSERS=1* to proceed without such a user:

```
[Actions]
DefVar $LOG_LOCATION$
Set $LOG_LOCATION$ = %opsiLogDir% + "\myproduct.log"
winbatch_install_myproduct

[winbatch_install_myproduct]
msiexec /qb ALLUSERS=1 /l* $LOG_LOCATION$ /i %SCRIPTPATH%\files\myproduct.msi
```

In other case it is necessary to create a temporary administrative user in whose context the installation takes place. This can be done as follows:

- Create a new product frame based on the product `opsi-template-with-admin`
- Create a directory *localsetup* in the product directory (i.e. in *install\productId*).
- Move all your installation files into the directory *CLIENT_DATA\localsetup*.
- Make sure that your setup script at *CLIENT_DATA\localsetup* starts with a reboot call:

```
[Actions]
ExitWindows /Reboot
```

- Edit at *CLIENT_DATA\setup.ins* the variables that are marked with: `Please edit the following values`.

The *opsi-winst/opsi-script* script template temporarily generates a user context, executes an installation in it, then removes it. Before using the template the following values are to be set adequately:

- the value for the variable *$Productname$*
- the value of the variable *$ProductSize$*
- the value of the variable `$LocalSetupScript$` (the name of your setup script)

The script proceeds as follows:

- It creates a local administrator opsiSetupAdmin;
- saves the autologon state;
- inserts opsiSetupAdmin as autologon user;
- copies the installation files to the client (as defined in $localFilesPath$); among them the installation script that is to be executed in the local user context;
- creates a RunOnce entry in the registry that calls *opsi-winst/opsi-script* with the local script as argument;
- reboots in order to make the registry change work;
- when *opsi-winst/opsi-script* runs again, it calls an ExitWindows /ImmediateLogout, and the second scripting level begins to work:
- By autologon , opsiSetupAdmin is logged on without user interaction.
- Windows calls the RunOnce command, that is the *opsi-winst/opsi-script* call.
- The *opsi-winst/opsi-script* script should now regularly proceed. But at its end, there must be a ExitWindows /ImmediateReboot command. Otherwise the desktop would of the administratrive user opsiSetupAdmin who is already logged at the moment would be accessible.
- after the reboot, the main script works again cleaning everything (writing back the old autologon state, deleting the local setup files, removing the opsiSetupAdmin profile)

We call the two involved *opsi-winst/opsi-script* scripts master script and local script . The first one runs in a system service context, the second which does the specific software installation runs in the context of a local administrator.

Caution
If the local script requires internal reboots then the master script must be adapted to produce them. As long as the local script is not finished the master script hands over control to the local script by an `ExitWindows /ImmediateLogout`. Of course the RunOnce entry has to be created for each run. Since username and password for the autologon are removed at the beginning of the local script they have to be reset each time as well.

There is (since opsi 4.0.2-2) a direct access from the local script to the product properties.

There may be product installations by external setup program calls which change registry entries which are saved by the master script and usually written back at the end of the installation. In this case the master script must be adapted to avoid writing back.

The local script runs with an administrator logged in. You have to lock the keyboard when testing is done. Otherwise anybody sitting at the client could stop script execution and take over the session. Therefore the product has a product property `debug` which switches input locking and log level.

In order to avoid logging of passwords the loglevel is temporarily set to -2.

Important
Please do not use the script as printed below, but use the opsi product: opsi-template-with-admin.

```
; Copyright (c) uib gmbh (www.uib.de)
; This sourcecode is owned by uib
; and published under the Terms of the General Public License.

; TEMPLATE for
; Skript fuer Installationen im Kontext eines temporaeren lokalen Administrators
; installations as temporary local admin
; see winst_manual.pdf / winst_handbuch.pdf

;
; !!! Das lokale Installations-Skript, das durch den temporaeren lokalen Admin ausgefuehrt wird
; !!! (sein Name steht in $LocalSetupScript$), muss mit dem Befehl
; !!! exitWindows /Reboot
; !!! enden
;

; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
; Vorarbeiten/Voraussetzungen/Doku pruefen wie in Winsthandbuch
;  Skript fuer Installationen im Kontext eines lokalen Administrators
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


[Actions]
requiredWinstVersion >= 4.11.4.12
setLogLevel=7
ScriptErrorMessages=off
DefVar $ProductName$
DefVar $ProductSizeMB$
DefVar $LocalSetupScript$
DefVar $LockKeyboard$
DefVar $OpsiAdminPass$
DefVar $RebootFlag$
DefVar $WinstRegKey$
DefVar $AutoName$
DefVar $AutoPass$
DefVar $AutoDom$
DefVar $AutoLogon$
DefVar $AutoBackupKey$
DefVar $LocalFilesPath$
DefVar $LocalWinst$
DefVar $DefaultLoglevel$
DefVar $PasswdLogLevel$
DefVar $AdminGroup$
DefVar $SearchResult$
DefVar $LocalDomain$
DefVar $debug$
DefVar $isFatal$

; ----------------------------------------------------------------
; - Please edit the following values
; ----------------------------------------------------------------
Set $ProductName$ = "opsi-template-with-admin"
Set $ProductSizeMB$ = "1"
Set $LocalSetupScript$ = "setup32.opsiscript"
; ----------------------------------------------------------------

comment "get and set initial values..."
set $debug$ = GetProductProperty("debug","false")
set $isFatal$ = "false"
set $DefaultLoglevel$ = "7"
SetLogLevel=$DefaultLoglevel$
Set $LocalDomain$ = "%PCNAME%"
```

```
comment "check if we productive or debugging..."
if $debug$ = "true"
        comment "we are in debug mode"
        Set $LockKeyboard$="false"
        Set $PasswdLogLevel$="7"
else
        comment "we are in productive mode"
        comment "set $LockKeyboard$ to true to prevent user hacks while admin is logged in"
        Set $LockKeyboard$="true"
        comment " set $PasswdLogLevel$ to 0 for production"
        Set $PasswdLogLevel$="0"
endif

comment "handle Rebootflag"
Set $WinstRegKey$ = "HKLM\SOFTWARE\opsi.org\winst\"+$ProductName$
Set $RebootFlag$ = GetRegistryStringValue32("["+$WinstRegKey$+"] "+"RebootFlag")

comment "some paths required"
Set $AutoBackupKey$ = $WinstRegKey$+"\AutoLogonBackup"
Set $LocalFilesPath$ = "C:\opsi.org\tmp\opsi_local_inst"
Set $LocalWinst$ = "%ProgramFilesDir%\opsi.org\opsi-client-agent\opsi-winst\winst32.exe"
if not( FileExists($LocalWinst$) )
        LogError "No opsi-winst found. Abborting."
        isFatalError
endif

comment "show product picture"
ShowBitmap "%scriptpath%\localsetup\"+$ProductName$+".png" $ProductName$

if not (($RebootFlag$ = "1") or ($RebootFlag$ = "2") or ($RebootFlag$ = "3"))
        comment "Part before first Reboot"
        comment "just reboot - this must be done if this is the first product after OS
    installation"
        comment "handle Rebootflag"
        Set $RebootFlag$ = "1"
        Registry_SaveRebootFlag /32bit
        ;ExitWindows /ImmediateReboot
endif ; Rebootflag = not (1 or 2 or 3)

if $RebootFlag$ = "1"
        comment "Part before second Reboot"
        setActionProgress "Preparing"

        if not(HasMinimumSpace ("%SYSTEMDRIVE%", ""+$ProductSizeMB$+" MB"))
                LogError "Not enough space on drive C: . "+$ProductSizeMB$+" MB on C: required
    for "+$ProductName$
                isFatalError
        endif

        comment "Lets work..."
        Message "Preparing "+$ProductName$+" install step 1..."
        sub_Prepare_AutoLogon

        comment "we need to reboot now to be sure that the autologon work"
        comment "handle Rebootflag"
        Set $RebootFlag$ = "2"
        Registry_SaveRebootFlag /32bit
        ExitWindows /ImmediateReboot
endif ; Rebootflag = not (1 or 2)

if ($RebootFlag$ = "2")
        comment "Part after first Reboot"

        comment "handle Rebootflag"
        Set $RebootFlag$ = "3"
        Registry_SaveRebootFlag /32bit

        comment "Lets work..."
        Message "Preparing "+$ProductName$+" install step 2..."
        Registry_enable_keyboard /sysnative

        comment "now let the autologon work"
        comment "it will stop with a reboot"
        setActionProgress "Run Installation"

        ExitWindows /ImmediateLogout
endif ; Rebootflag = 2

if ($RebootFlag$ = "3")
        comment "Part after second Reboot"
        ExitWindows /Reboot
        setActionProgress "Cleanup"
        comment "handle Rebootflag"
        Set $RebootFlag$ = "0"
        Registry_SaveRebootFlag /32bit

        comment "Lets work..."
        Message "Cleanup "+$ProductName$+" install (step 3)..."
        sub_Restore_AutoLogon
        set $SearchResult$ = GetRegistryStringValueSysnative("[HKLM\SOFTWARE\Microsoft\Windows\
    CurrentVersion\RunOnce] opsi_autologon_setup")
        if $SearchResult$ = $LocalWinst$+" "+$LocalFilesPath$+"\"+$LocalSetupScript$+" /batch /
    productid %installingProdName%"
                LogError "Localscript did not run. We remove the RunOnce entry and abort"
                Registry_del_runonce /sysnative
                set $isFatal$ = "true"
        endif
        if "true" = getRegistryStringValue32("[HKLM\Software\opsi.org\winst] with-admin-fatal")
                LogError "set to fatal because the local script stored this result"
                set $isFatal$ = "true"
        endif
        comment "cleanup the registry key which stores a fatal result of the local script"
        Registry_clean_fatal_flag /32bit
        if $isFatal$ = "true"
                isFatalError
        endif
        comment "This is the clean end of the installation"
endif ; Rebootflag = 3


[sub_Prepare_AutoLogon]
comment "copy the setup script and files"
Files_copy_Setup_files_local
comment "read actual Autologon values for backup"
```

```
set $AutoName$ = GetRegistryStringValueSysnative("[HKLM\SOFTWARE\Microsoft\Windows NT\
    CurrentVersion\Winlogon] DefaultUserName")
comment "if AutoLogonName is our setup admin user, something bad happend"
comment "then let us cleanup"
if ($AutoName$="opsiSetupAdmin")
        set $AutoName$=""
        set $AutoPass$=""
        set $AutoDom$=""
        set $AutoLogon$="0"
else
        set $AutoPass$ = GetRegistryStringValueSysnative("[HKLM\SOFTWARE\Microsoft\Windows NT\
    CurrentVersion\Winlogon] DefaultPassword")
        set $AutoDom$ = GetRegistryStringValueSysnative("[HKLM\SOFTWARE\Microsoft\Windows NT\
    CurrentVersion\Winlogon] DefaultDomainName")
        set $AutoLogon$ = GetRegistryStringValueSysnative("[HKLM\SOFTWARE\Microsoft\Windows NT\
    CurrentVersion\Winlogon] AutoAdminLogon")
endif

comment "backup AutoLogon values"
Registry_save_autologon /32bit

comment "prepare the admin AutoLogon"
SetLogLevel=$PasswdLogLevel$
set $OpsiAdminPass$= randomstr
Registry_autologon /sysnative

comment "get the name of the admin group"
set $AdminGroup$ = SidToName("S-1-5-32-544")
comment "create our setup admin user"
DosInAnIcon_makeadmin
SetLogLevel=$DefaultLoglevel$

comment "store our setup script as run once"
Registry_runOnce /sysnative

comment "disable keyboard and mouse while the autologin admin works"
if ($LockKeyboard$="true")
        Registry_disable_keyboard /Sysnative
endif

comment "cleanup the registry key which stores a fatal result of the local script"
Registry_clean_fatal_flag /32bit

[sub_Restore_AutoLogon]
comment "read AutoLogon values from backup"
set $AutoName$ = GetRegistryStringValue("["+$AutoBackupKey$+"] DefaultUserName")
set $AutoPass$ = GetRegistryStringValue("["+$AutoBackupKey$+"] DefaultPassword")
set $AutoDom$ = GetRegistryStringValue("["+$AutoBackupKey$+"] DefaultDomainName")
set $AutoLogon$ = GetRegistryStringValue("["+$AutoBackupKey$+"] AutoAdminLogon")

comment "restore the values"
SetLogLevel = $PasswdLogLevel$
Registry_restore_autologon /Sysnative
SetLogLevel = $DefaultLoglevel$
comment "delete our setup admin user"
DosInAnIcon_deleteadmin
comment "cleanup setup script, files and profiledir"
```

```
Files_delete_Setup_files_local
comment "delete profiledir"
DosInAnIcon_deleteprofile

[Registry_save_autologon]
openkey [$AutoBackupKey$]
set "DefaultUserName"="$AutoName$"
set "DefaultPassword"="$AutoPass$"
set "DefaultDomainName"="$AutoDom$"
set "AutoAdminLogon"="$AutoLogon$"

[Registry_restore_autologon]
openkey [HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon]
set "DefaultUserName"="$AutoName$"
set "DefaultPassword"="$AutoPass$"
set "DefaultDomainName"="$AutoDom$"
set "AutoAdminLogon"="$AutoLogon$"

[DosInAnIcon_deleteadmin]
NET USER opsiSetupAdmin  /DELETE

[Registry_SaveRebootFlag]
openKey [$WinstRegKey$]
set "RebootFlag" = "$RebootFlag$"

[Files_copy_Setup_files_local]
copy -s "%ScriptPath%\localsetup\*.*" "$LocalFilesPath$"

[Files_delete_Setup_files_local]
del -sf $LocalFilesPath$\
; folgender Befehl funktioniert nicht vollständig, deshalb ist er zur Zeit auskommentier
; der Befehl wird durch die Sektion "DosInAnIcon_deleteprofile" ersetzt (P.Ohler)
;delete -sf "%ProfileDir%\opsiSetupAdmin"

[DosInAnIcon_deleteprofile]
rmdir /S /Q "%ProfileDir%\opsiSetupAdmin"

[DosInAnIcon_makeadmin]
NET USER opsiSetupAdmin $OpsiAdminPass$ /ADD
NET LOCALGROUP $AdminGroup$ /ADD opsiSetupAdmin

[Registry_autologon]
openkey [HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon]
set "DefaultUserName"="opsiSetupAdmin"
set "DefaultPassword"="$OpsiAdminPass$"
set "DefaultDomainName"="$LocalDomain$"
set "AutoAdminLogon"="1"

[Registry_runonce]
openkey [HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce]
set "opsi_autologon_setup"='"$LocalWinst$" "$LocalFilesPath$\$LocalSetupScript$" /batch /
    productid %installingProdName%'

[Registry_del_runonce]
openkey [HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce]
DeleteVar "opsi_autologon_setup"
```

```
[Registry_disable_keyboard]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kbdclass]
set "Start"=REG_DWORD:0x4
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Mouclass]
set "Start"=REG_DWORD:0x4

[Registry_enable_keyboard]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kbdclass]
set "Start"=REG_DWORD:0x1
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Mouclass]
set "Start"=REG_DWORD:0x1

[Registry_clean_fatal_flag]
openkey [$WinstRegKey$]
DeleteVar "with-admin-fatal"
```

12.4 XML File Patching: Setting Template Path for OpenOffice.org 2

Setting the template path can be done by the following script extracts

```
[Actions]
; ....

DefVar $oooTemplateDirectory$
;-----------------------------------------------------------
;set path here:

Set $oooTemplateDirectory$ = "file://server/share/verzeichnis"
;-----------------------------------------------------------
;...

DefVar $sofficePath$
Set $sofficePath$= GetRegistryStringValue ("[HKEY_LOCAL_MACHINE\SOFTWARE\OpenOffice.org\
    OpenOffice.org\2.0] Path")
DefVar $oooDirectory$
Set $oooDirectory$= SubstringBefore ($sofficePath$, "\program\soffice.exe")
DefVar $oooShareDirectory$
Set $oooShareDirectory$ = $oooDirectory$ + "\share"

XMLPatch_paths_xcu $oooShareDirectory$+"\registry\data\org\openoffice\Office\Paths.xcu"
; ...


[XMLPatch_paths_xcu]
OpenNodeSet
- error_when_no_node_existing false
- warning_when_no_node_existing true
- error_when_nodecount_greater_1 false
- warning_when_nodecount_greater_1 true
- create_when_node_not_existing true
- attributes_strict false

documentroot
all_childelements_with:
elementname: "node"
```

```
attribute:"oor:name" value="Paths"
all_childelements_with:
elementname: "node"
attribute: "oor:name" value="Template"
all_childelements_with:
elementname: "node"
attribute: "oor:name" value="InternalPaths"
all_childelements_with:
elementname: "node"

end

SetAttribute "oor:name" value="$oooTemplateDirectory$"
```

12.5 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 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:

```
[XMLPatch_db_config]
openNodeSet
        documentroot
        all_childelements_with:
                elementname:"appSettings"
        all_childelements_with:
                elementname:"add"
                attribute: "key" value ="Database.DataSource"
end
SetAttribute "value" value="$source$"

openNodeSet
        documentroot
        all_childelements_with:
                elementname:"appSettings"
        all_childelements_with:
                elementname:"add"
                attribute: "key" value ="Database.InitialCatalog"
end
```

```
SetAttribute "value" value="$catalog$"
```

12.6 Retrieving Values From a XML File

As treated in Section 12.4 , *opsi-winst/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 version="1.0" encoding="utf-16" ?>
<Collector xmlns="http://schemas.microsoft.com/appx/2004/04/Collector" xmlns:xs="http://www.w3.
    org/2001/XMLSchema-instance" xs:schemaLocation="Collector.xsd" UtcDate="04/06/2006 12:28:17"
    LogId="{693B0A32-76A2-4FA0-979C-611DEE852C2C}"  Version="4.1.3790.1641" >
   <Options>
      <Department></Department>
      <IniPath></IniPath>
      <CustomValues>
      </CustomValues>
   </Options>
   <SystemList>
      <ChassisInfo Vendor="Chassis Manufacture" AssetTag="System Enclosure 0" SerialNumber="EVAL"
    />
      <DirectxInfo Major="9" Minor="0"/>
   </SystemList>
   <SoftwareList>
      <Application Name="Windows XP-Hotfix - KB873333" ComponentType="Hotfix" EvidenceId="256"
    RootDirPath="C:\WINDOWS\$NtUninstallKB873333$\spuninst" OsComponent="true" Vendor="Microsoft
    Corporation" Crc32="0x4235b909">
         <Evidence>
            <AddRemoveProgram DisplayName="Windows XP-Hotfix - KB873333" CompanyName="Microsoft
    Corporation" Path="C:\WINDOWS\$NtUninstallKB873333$\spuninst" RegistryPath="
    HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KB873333"
    UninstallString="C:\WINDOWS\$NtUninstallKB873333$\spuninst\spuninst.exe" OsComponent="true"
    UniqueId="256"/>
         </Evidence>
      </Application>
      <Application Name="Windows XP-Hotfix - KB873339" ComponentType="Hotfix" EvidenceId="257"
    RootDirPath="C:\WINDOWS\$NtUninstallKB873339$\spuninst" OsComponent="true" Vendor="Microsoft
    Corporation" Crc32="0x9c550c9c">
         <Evidence>
            <AddRemoveProgram DisplayName="Windows XP-Hotfix - KB873339" CompanyName="Microsoft
    Corporation" Path="C:\WINDOWS\$NtUninstallKB873339$\spuninst" RegistryPath="
    HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KB873339"
    UninstallString="C:\WINDOWS\$NtUninstallKB873339$\spuninst\spuninst.exe" OsComponent="true"
    UniqueId="257"/>
         </Evidence>
      </Application>
   </SoftwareList>
</Collector>
```

To read the elements and get the values of all „Application“ nodes we may use these extracts of code:

```
[Actions]
DefStringList $list$

...
```

```
set $list$ = getReturnListFromSection ('XMLPatch_findProducts '+$TEMP$+'\test.xml')
for $line$ in $list$ do Sub_doSomething

[XMLPatch_findProducts]
openNodeSet
        ; Node „Collector" is  documentroot
        documentroot
        all_childelements_with:
          elementname:"SoftwareList"
        all_childelements_with:
          elementname:"Application"
end
return elements

[Sub_doSomething]
set $escLine$ = EscapeString:$line$
; now we can work on the content of $escLine$
```

We encapsulate the retrieved Strings by setting their values as a whole into an 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 *opsi-winst/opsi-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 opsiclientd determines and knows which event is currently active. `opsi-script` can be used by means of an *opsiservicecall* And thus connect with the *opsiclientd* querying the corresponding events:

```
[actions]
setLogLevel=5
DefVar $queryEvent$
DefVar $result$

;==================================
set $queryEvent$ = "gui_startup"

set serviceInfo = getReturnListFromSection('opsiservicecall_event_on_demand_is_running /
    opsiclientd')
set $result$ = takestring(0, serviceInfo)
if $result$ = "true"
        comment "event " + $queryEvent$ + " is running"
else
        comment "NOT running event " + $queryEvent$
endif

;==================================
set $queryEvent$ = "on_demand"

set serviceInfo = getReturnListFromSection('opsiservicecall_event_on_demand_is_running /
    opsiclientd')
set $result$ = takestring(0, serviceInfo)
if $result$ = "true"
        comment "event " + $queryEvent$ + " is running"
else
        comment "NOT running event " + $queryEvent$
endif

;==================================
set $queryEvent$ = "on_demand{user_logged_in}"

set serviceInfo = getReturnListFromSection('opsiservicecall_event_on_demand_is_running /
    opsiclientd')
set $result$ = takestring(0, serviceInfo)
if $result$ = "true"
        comment "event " + $queryEvent$ + " is running"
else
        comment "NOT running event " + $queryEvent$
endif
```

Chapter 13

Special Error Messages

- No Connection with the opsi Service
  The *opsi-winst/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 documentations in asciidoc format from the following sources:

- opsi-script library files (opsi-script)
- opsi webservice interface definition files (python) (implementation at work)

Why asciidoc as output ?

- asciidoc is the standard format for all opsi documentatation
- asciidoc is a base format to create documents in different formats like `html, pdf, epub, docbook, ...`

The documentation is created from informations, that are extracted from the source code. From the source code opsi-doc-generator knows the definitions of opsi-script defined functions and can get the informazion from there.
For additional information there may be special markers in comment lines which hold additional information on the level of file, function and parameter.

14.2 opsi-doc-generator program

There are two variants of this program:

- The GUI Version `opsi-doc-generator-gui`:
- The CLI Version `opsi-doc-generator`:

```
$ ./opsi_doc_generator --help
Creates asciidoc from commented 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 programs for Linux and Windows as opsi packages in the contribute area on download.uib.de

14.3 opsi-doc-generator marker

There are three different levels where information can be found in a source file:

- file
- function (may be more than one in a file)
- function parameter (may be more than one in a function)

Every marker starts with the language specific comment char (opsiscript=;) followed by the the @ char and the marker identifier string.

Every allowed marker can occour never, once or multiple time on a level. If a marker occour multiple times, all lines of this marker are concatenated

After a marker one or more space chars have to be used before the information start

OPSI-SCRIPT MARKERS ALLOWED ON THE FILE LEVEL

- `;@author`
- `;@email`
- `;@date`
- `;@copyright`
- `;@version`
- `;@filedesc` Description of file

OPSI-SCRIPT MARKERS ALLOWED ON THE FUNCTION LEVEL

- `;@author` Author (if absent author of file is used)
- `;@email` eMail address (if absent eMail of file is used)
- `;@date` Date (if absent date of file is used)
- `;@copyright` copyright (if absent copyright of file is used)
- `;@version` version (if absent version of file is used)
- `;@Description` Description of function
- `;@Returns` Return value of function
- `;@OnError` What happens in the case of an error
- `;@SpecialCase` What happens in known special unexpected cases like empty input, no network, and so son
- `;@Requires`
- `;@References` The name of an other function in this file that are related to this function. only one per line. For multiple references use multiple lines
- `;@Links`
- `;@Example` An example for the use of this function.
  Examples are in most cases multiline with idents. The start of the information in the first example line defines the base ident. Idents have to be done with space chars only (no tabs).

OPSI-SCRIPT MARKERS ALLOWED ON THE FUNCTION PARAMETER LEVEL

- `;@ParamDesc_<praram name>` Description of the parameter <praram name>
- `;@ParamAdvice_<praram name>` Advice to the parameter <praram name>. That may be for example restrictions for valid values.

14.4 opsi-doc-generator examples

opsi-script markers on the file level

```
;@author           detlef oertel
;@email            d.oertel@uib.de
;@date             17.4.2018
;@copyright        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

```
;@author           detlef oertel
;@date             17.5.2018
;@Description      Sets for the given list of opsi productIds the action request
;@Description      to 'setup' (also resolving the dependencies)
;@Returns          Returns string "true" if all is ok
;@OnError          Returns string "false"
;@SpecialCase      Works only in opsi service mode (not in interactive or batch mode)
;@References
;@Links
;@ParamDesc_$productlist$     List of opsi product Ids
;@ParamAdvice_$productlist$
;@Example     [actions]
;@Example     DefStringlist $productlist$
;@Example
;@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-winst Tutorial (1.0.0)

15.1 Introduction

This tutorial should help you to learn some advanced features (e.g. string lists) of the opsi-winst script language.

Before we start some hints:

- you should always use opsi script constants if they aplicable. For example use *%system%* instead of *c:\windows\system32*.
- You shold use the opsi-winst manuals for further description of the mentioned script commands:
  - `opsi-winst manual`
  - `opsi-winst reference card`
- You should use the opsi product *opsi-script-test* as a running reference script which is calling (nearly) every opsi-winst command.

15.2 Creating opsi-winst scripts

You may use every text editor. We recommend to use the jedit editor with integrated opsi-winst syntax highlighting.

For testing opsi-winst scripts it is a good idea to run them from an interactive started opsi-winst. (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-winst 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-winst 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-winst 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-winst 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-winst functions:

- `DosInAnIcon winst /64bit`

15.3 Solutions

15.3.1 Solution Lection 1

```
[Actions]
comment "Show all Systemfiles"
DosInAnIcon_Dir


[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir
```

15.3.2 Solution Lection 2

```
[Actions]
DefStringList $list1$

comment "Show all Systemfiles"
comment "Output from DosInAnIcon is assingned to a list"
set $list1$ = getOutStreamFromSection ("DosInAnIcon_Dir")

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir
```

15.3.3 Solution Lection 3

```
[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(<list>,<search string>)
;get a partial list with all strings that match <search string>
comment "list with only DDL-Files"
set $list1$ = getlistContaining ($list1$,".dll")
comment "Number of DDL-Files"
set $DLLCount$ = count ($list1$)
comment "Number of DLL-Files: " + $DLLCount$

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir *.*
```

15.3.4 Solution Lection 4

```
[Actions]
setloglevel = 7
DefVar $dirline$

DefStringList $list1$

comment "Show all Systemfiles"
;DosInAnIcon_Dir
comment "Output from DosInAnIcon is setting to a list"
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 $list1$ = splitStringOnWhiteSpace($dirline$)
        set $dirline$ = takeString (2,$list1$)
        comment "Size of Kernel32.dll: "+$dirline$+" B"
endif

[DosInAnIcon_Dir]
%systemdrive%
cd %system%
dir *.*
```

15.3.5 Solution Lection 5

```
[Actions]
setloglevel = 7
DefVar $dirline$
DefVar $dirline64$
DefStringList $list32$
DefStringList $list64$

;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 ist " + $dirline$ + " 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-winst and are mantained by the uib gmbh

16.1.1 Documentation of opsi library: `uib_backend.opsiscript`

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

16.1.1.1 Documentation of local function `setProductsToSetup`

Definition
`setProductsToSetup($productlist$ : stringlist) : string`

Description
Sets for the given list of opsi productIds the action request to *setup* (also resolving the dependencies)

- Parameter: `$productlist$`
  - Type: `Stringlist` - Calltype: `CallByValue`
  - Parameter `$productlist$` Description:
    List of opsi product Ids
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
[actions]
DefStringlist $productlist$

set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
if not(stringtobool(setProductsToSetup($productlist$)))
  comment "call of setProductsToSetup failed"
endif
```

16.1.1.2 Documentation of local function `getInstalledLocalbootProducts`

Definition
`getInstalledLocalbootProducts(ref $productlist$ : stringlist) : string`

Description
Gets a list of productIds which are

- known to the client (productOnClient object exists)
- and localboot products. to *setup* (also resolving the dependencies)
- Parameter: `$productlist$`
  - Type: `Stringlist` - Calltype: `CallByReference`
  - Parameter `$productlist$` Description:
    Output list of opsi product Ids that were be found
  - Parameter `$productlist$` Advice:
    May be empty
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 20.4.2018
- Email: d.oertel@uib.de
- Version: 1.1
- Copyright: AGPLv3

Example:

```
if stringToBool(getInstalledLocalbootProducts($resultlist$))
        comment "getInstalledLocalbootProducts successful finished"
else
        LogError "getInstalledLocalbootProducts failed"
endif
set $tmplist$ = getListContainingList($baseproducts$,$resultlist$)
if stringToBool(compareLists($tmplist$,$baseproducts$))
        comment "check installed products successful finished"
else
        LogError "check installed products failed"
endif
comment " now install and rest products ...."
if stringToBool(setProductsToSetup($resetproducts$))
        comment "setProductsToSetup successful finished"
else
        LogError "setProductsToSetup failed"
endif
```

16.1.1.3 Documentation of local function `delOpsiPoc`

Definition
`delOpsiPoc($donotdelList$ : stringlist) : string`

Description
Delete all productOnClientObjects for this client and only for localboot products and not for products that are included in the $donotdelList$ parameter

- Parameter: `$donotdellist$`
  - Type: `Stringlist` - Calltype: `CallByValue`
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
set $opsiMetaDataFile$ = $targetDir$+"\poc.json"
if Fileexists($opsiMetaDataFile$)
        comment "Delete existing meta data ...."
        set $tmplist$ = createStringList ("opsi-vhd-tester","opsi-vhd-control","opsi-vhd-auto-
    upgrade")
        if stringToBool(delOpsiPoc($tmplist$))
                comment "Delete existing meta data successful finished"
        else
                LogError "Delete existing meta data failed"
        endif
        comment "Restore existing meta data ...."
        if stringToBool(restoreOpsiPoc($opsiMetaDataFile$))
                comment "Restore existing meta data successful finished"
        else
                LogError "Restore existing meta data failed"
        endif
else
        comment "No meta data existing - creating it...."
        if stringToBool(backupOpsiPoc($opsiMetaDataFile$))
                comment "Backup meta data successful finished"
        else
                LogError "Backup meta data failed"
        endif
endif
```

16.1.1.4 Documentation of local function `backupOpsiPoc`

Definition
`backupOpsiPoc($filename$ : string) : string`

Description
Get all localboot productOnClient objects for this client and write it to the json file $filename$

- Parameter: `$filename$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$filename$` Description:
    Complete name of the file to create
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- References: [restoreOpsiPoc] [delOpsiPoc]
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
see delOpsiPoc
```

16.1.1.5 Documentation of local function `restoreOpsiPoc`

Definition
`restoreOpsiPoc($filename$ : string) : string`

Description
Load productOnClient objects from $filename$ and write it to to the server

- Parameter: `$filename$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$filename$` Description:
    Complete name of the file to read
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
see delOpsiPoc
```

16.1.1.6 Documentation of local function `getInstalledLocalbootProductsWithVersion`

Definition
`getInstalledLocalbootProductsWithVersion(ref $productlist$ : stringlist) : string`

Description
Get all localboot productOnClient objects for this client and create a key/value list in the format <productId>=<productVersion>-<packageVersion> This list is be written to $productlist$

- Parameter: `$productlist$`
  - Type: `Stringlist` - Calltype: `CallByReference`
  - Parameter `$productlist$` Description:
    The key/value list with all <productId>=<productVersion>-<packageVersion> of the client
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
comment "check for installed products ...."
if stringToBool(getInstalledLocalbootProductsWithVersion($installedproducts$))
        comment "getInstalledLocalbootProducts successful finished"
else
        LogError "getInstalledLocalbootProducts failed"
endif
comment "check for installable products ...."
if stringToBool(getInstallableLocalbootProductsWithVersion($possibleproducts$))
        comment "getInstallableLocalbootProductsWithVersion successful finished"
else
        LogError "getInstallableLocalbootProductsWithVersion failed"
endif
comment "fill $upgradeproducts$ .."
set $tmplist$ = getKeyList($installedproducts$)
set $possibleproducts$ = getSubListByKey($tmplist$,$possibleproducts$)
for %aktprod% in $installedproducts$ do sub_find_updatable_products
```

16.1.1.7 Documentation of local function `getInstallableLocalbootProductsWithVersion`

Definition
`getInstallableLocalbootProductsWithVersion(ref $productlist$ : stringlist) : string`

Description
Get all localboot productOnDepot objects for the depot of this client and create a key/value list in the format <productId>=<productVersion>-<packageVersion> This list is be written to $productlist$

- Parameter: `$productlist$`
  - Type: `Stringlist` - Calltype: `CallByReference`

  - Parameter `$productlist$` Description:
    The key/value list with all <productId>=<productVersion>-<packageVersion> of the depot of this client
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Works only in opsi service mode (not in interactive or batch mode)
- Author: detlef oertel
- Date: 17.4.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
see getInstalledLocalbootProductsWithVersion
```

16.1.2 Documentation of opsi library: `uib_bootutils.opsiscript`

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

16.1.2.1 Documentation of local function `delFromWindowsBootmanager`

Definition
`delFromWindowsBootmanager($bootlabel$ : string) : string`

Description
Deletes the boot entry given given by the parameter $bootlabel$ from Windows boot manager by using bcdedit

- Parameter: `$bootlabel$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$bootlabel$` Description:
    Windows boot manager entry label found in *bcdedit /v*
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [getWinBcdbootGuid] [bootNextToWinLabel] [bootNextToUefiLabel] [getUefiBcdbootGuid]
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
[actions]
DefStringlist $productlist$

set $productlist$ = CreateStringList("opsi-logviewer","opsi-configed")
if not(stringtobool(setProductsToSetup($productlist$)))
  comment "call of setProductsToSetup failed"
endif
```

16.1.2.2 Documentation of local function `getWinBcdbootGuid`

Definition
`getWinBcdbootGuid($bootlabel$ : string) : string`

Description
Get the boot entry GUID for the label given by the parameter $bootlabel$ from Windows boot manager by using bcdedit

- Parameter: `$bootlabel$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$bootlabel$` Description:
    Windows boot manager entry label found in *bcdedit /v*
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [delFromWindowsBootmanager] [bootNextToWinLabel] [bootNextToUefiLabel] [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")
```

16.1.2.3 Documentation of local function `getUefiBcdbootGuid`

Definition
`getUefiBcdbootGuid($bootlabel$ : string) : string`

Description
Get the boot entry GUID for the label given by the parameter $bootlabel$ from Windows boot manager by using bcdedit

- Parameter: `$bootlabel$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$bootlabel$` Description:
    UEFI boot manager entry label found in *bcdedit /enum firmware*
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [delFromWindowsBootmanager] [bootNextToWinLabel] [bootNextToUefiLabel] [getUefiBcdbootGuid]
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de

- Version: 1.0
- Copyright: AGPLv3

Example:

```
if runningonUefi
        set $peuefiguid$ = getUefiBcdbootGuid("opsitempwinpe")
        set $exitcode$ = getlastexitcode
        if $exitcode$ = "0"
                if not ($peuefiguid$ = "")
                        shellCall("bcdedit /delete "+$peuefiguid$)
                endif
        endif
endif
```

16.1.2.4 Documentation of local function `bootNextToWinLabel`

Definition
`bootNextToWinLabel($bootlabel$ : string) : string`

Description
Sets the Windows bootmanager to boot next to the label given by the parameter $bootlabel$ from Windows boot manager by using bcdedit

- Parameter: `$bootlabel$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$bootlabel$` Description:
    Windows boot manager entry label found in *bcdedit /v*
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [delFromWindowsBootmanager] [getUefiBcdbootGuid] [bootNextToUefiLabel]
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
See bootNextToUefiLabel
```

16.1.2.5 Documentation of local function `bootNextToUefiLabel`

Definition
`bootNextToUefiLabel($bootlabel$ : string) : string`

Description
Sets the uefi bootmanager to boot next to the label given by the parameter $bootlabel$ from uefi boot manager by using bcdedit

- Parameter: `$bootlabel$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$bootlabel$` Description:
    UEFI boot manager entry label found in *bcdedit /enum firmware*
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [delFromWindowsBootmanager] [getUefiBcdbootGuid] [bootNextToUefiLabel]
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
Message "Enable PE boot...."
if runningonuefi
        set $bootLabel$ = "opsitempwinpe"
        if not(stringToBool(bootNextToUefiLabel($bootLabel$)))
                logerror "Activating peboot is failed"
                isFatalError "failed peboot"
        endif
else
        set $bootLabel$ = "ramdisk=[boot]\sources\boot.wim"
        if not(stringToBool(bootNextToWinLabel($bootLabel$)))
                logerror "Activating peboot is failed"
                ;isFatalError "failed peboot"
        endif
endif
```

16.1.2.6 Documentation of local function `getDiskUuid`

Definition
`getDiskUuid($disknumber$ : string , $tmpdir$ : string ) : string`

Description
Gets the disk uuid for the disk with the number $disknumber$ by using diskpart. The temporary diskpart script is written to $tmpdir$

- Parameter: `$disknumber$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$disknumber$` Description:
    UEFI boot manager entry label found in *bcdeit /enum firmware*
- Parameter: `$tmpdir$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$tmpdir$` Description:
    Temporary directory to use
  - Parameter `$tmpdir$` Advice:
    Directory must exist
- Returns: Returns string "true" if all is ok

- OnError: Returns string "false"
- SpecialCase: Winows only
- References: [enablePEPartition]
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
for %disk% = "0" to calculate($diskcount1$+" -1") do set $aktdisklist$ = addtolist($aktdisklist$,
    getDiskUuid("%disk%", "x:")+"=%disk%")
set $disk$ = getvalue($diskuuid$,$aktdisklist$)
```

16.1.2.7 Documentation of local function `enablePEPartition`

Definition
`enablePEPartition($disknumber$ : string , $partitionNumber$ : string, $pepartletter$ : string, $useGpt$ : string) : string`

Description
Try to make the partition $partitionNumber$ on the disk $disknumber$ visible, bootable and give and give it the Windows disk letter $pepartletter$ by using diskpart or powershell

- Parameter: `$disknumber$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$disknumber$` Description:
    Number of the disk where we look for the partition
  - Parameter `$disknumber$` Advice:
    First disk = 0
- Parameter: `$partitionnumber$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$partitionnumber$` Description:
    Number of the partition on the given disk
  - Parameter `$partitionnumber$` Advice:
    First partition = 1
- Parameter: `$pepartletter$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$pepartletter$` Description:
    Windows disk letter that the given partition should have
- Parameter: `$usegpt$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$usegpt$` Description:
    Should we expect GPT or MBR partitions (*true* or *false*)
- Returns: Returns string "true" if all is ok
- OnError: Returns string "false"
- SpecialCase: Winows only, works in PE
- References: [getDiskUuid]

- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
Message "Enable PE partition...."
if not(stringToBool(enablePEPartition($disknumber$, $swapPartitionNumber$, $pepartletter$, $
    useGpt$)))
        LogError "Could not activate PE partition ..."
        isFatalError "failed not activate PE partition"
endif

if not (isDriveReady($pePartLetter$))
        logerror "PE drive "+$pePartLetter$+": not ready"
        isFatalError "PE drive "+$pePartLetter$+": not ready"
        set $errorList$ = addtolist($errorList$, " failed pe_drive_ready")
        set $fatal_error$ = "true"
endif
```

16.1.3 Documentation of opsi library: `uib_exitcode.opsiscript`

- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.1 Documentation of local function `isMsiExitcodeFatal`

Definition

`isMsiExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string`

Description

Evaluates the given $exitcode$ as MSI Error and 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: Winows only
- References: [isMsExitcodeFatal_short] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshield-ExitcodeFatal] [isNsisExitcodeFatal]
- Links: http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx http://msdn.microsoft.com/en-us/-library/aa368542.aspx
- Author: detlef oertel
- Date: 19.9.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

Example:

```
[actions]
DefVar $ExitCode$
DefVar $ErrorString$
(...)
set $ExitCode$ = getlastexitcode
if stringtobool(isMsiExitcodeFatal($exitcode$, "true", $ErrorString$ ))
        LogError $ErrorString$
        isfatalerror $ErrorString$
else
        Comment $ErrorString$
endif
```

16.1.3.2 Documentation of local function `isMsExitcodeFatal_short`

Definition

`isMsExitcodeFatal_short($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string`

Description

Evaluates the given $exitcode$ as MS Error and 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.
- OnError: Returns string "true"
- SpecialCase: Winows only
- References: [isMsiExitcodeFatal] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExit-codeFatal] [isNsisExitcodeFatal]
- Links: http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx http://msdn.microsoft.com/en-us/-library/aa368542.aspx
- Author: detlef oertel
- Date: 19.9.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.3 Documentation of local function `isAdvancedMsiExitcodeFatal`

Definition

`isAdvancedMsiExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string`

Description

Please note: Import complete file uib_exitcode (not only isAdvancedMsiExitcodeFatal) Evaluates the given $exitcode$ as AdvancedMsi Error and and gives back a resulting error message on $ErrorString$ It is also checked if the exit code is one from the embedded msi. There for is isMsiExitcodeFatal used If the Error require a reboot the given parameter $allowRebootRequest$ is used to decide if a *ExitWindows /Reboot* is called or not

- Parameter: `$exitcode$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$exitcode$` Description:
    Exit code given by AdvancedMsi
- Parameter: `$allowrebootrequest$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$allowrebootrequest$` Description:
    Should we call *ExitWindows /Reboot* if the exit code require this (*true* or *false*)
- Parameter: `$errorstring$`
  - Type: `String` - Calltype: `CallByReference`
  - Parameter `$errorstring$` Description:
    Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"
- SpecialCase: Winows only
- References: [isMsiExitcodeFatal] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExit-codeFatal] [isNsisExitcodeFatal]
- Author: detlef oertel
- Date: 19.9.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.4 Documentation of local function `isInnoExitcodeFatal`

Definition

`isInnoExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string`

Description

Evaluates the given $exitcode$ as Inno Error and and gives back a resulting error message on $ErrorString$ If the Error require a reboot the given parameter $allowRebootRequest$ is used to decide if a *ExitWindows /Reboot* is called or not

- Parameter: `$exitcode$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$exitcode$` Description:
    Exit code given by Inno
- Parameter: `$allowrebootrequest$`
  - Type: `String` - Calltype: `CallByValue`
  - Parameter `$allowrebootrequest$` Description:
    Should we call *ExitWindows /Reboot* if the exit code require this (*true* or *false*)
- Parameter: `$errorstring$`
  - Type: `String` - Calltype: `CallByReference`
  - Parameter `$errorstring$` Description:
    Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"
- SpecialCase: Winows only
- References: [isMsiExitcodeFatal] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExitcodeFatal] [isNsisExitcodeFatal]
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.5 Documentation of local function `isInstallshieldExitcodeFatal`

Definition

`isInstallshieldExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string`

Description

Please note: Import complete file uib_exitcode (not only isAdvancedMsiExitcodeFatal) Evaluates the given $exitcode$ as Installshield Error and 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: Winows only
- References: [isMsiExitcodeFatal] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExitcodeFatal] [isNsisExitcodeFatal]
- Author: detlef oertel
- Date: 19.9.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.3.6 Documentation of local function `isNsisExitcodeFatal`

Definition
`isNsisExitcodeFatal($exitcode$ : string, $allowRebootRequest$ : string, ref $ErrorString$ : string) : string`

Description
Evaluates the given $exitcode$ as Nsis Error and and gives back a resulting error message on $ErrorString$ If the Error require a reboot the given parameter $allowRebootRequest$ is used to decide if a *ExitWindows /Reboot* is called or not

- Parameter: `$exitcode$`
    - Type: `String` - Calltype: `CallByValue`
    - Parameter `$exitcode$` Description:
      Exit code given by Nsis
- Parameter: `$allowrebootrequest$`
    - Type: `String` - Calltype: `CallByValue`
    - Parameter `$allowrebootrequest$` Description:
      Should we call *ExitWindows /Reboot* if the exit code require this (*true* or *false*)
- Parameter: `$errorstring$`
    - Type: `String` - Calltype: `CallByReference`
    - Parameter `$errorstring$` Description:
      Here we get the error string that belongs to the given exit code
- Returns: Returns string "true" if the exit code points to a critical error. Returns string "true" if the exit code is not a number (not valid). Returns string "false" if the exit code is valid but not critical.
- OnError: Returns string "true"

- SpecialCase: Winows only
- References: [isMsiExitcodeFatal] [isAdvancedMsiExitcodeFatal] [isInnoExitcodeFatal] [isInstallshieldExitcodeFatal] [isNsisExitcodeFatal]
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0.1
- Copyright: AGPLv3

16.1.4 Documentation of opsi library: `uib_strlistutils.opsiscript`

- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

16.1.4.1 Documentation of local function `compareLists`

Definition
`compareLists($list1$ : stringlist, $list2$ : stringlist) : string`

Description
Checks if to string lists are completely identic or not. The check is not case sensitive.

- Parameter: `$list1$`
  - Type: `Stringlist` - Calltype: `CallByValue`
  - Parameter `$list1$` Description:
    First stringlist to compare
- Parameter: `$list2$`
  - Type: `Stringlist` - Calltype: `CallByValue`
  - Parameter `$list2$` Description:
    Second stringlist (to compare with first)
- Returns: Returns string "true" if the given lists are identic. Returns string "false" if the given lists are not identic
- OnError: Returns string "false"
- Author: detlef oertel
- Date: 17.5.2018
- Email: d.oertel@uib.de
- Version: 1.0
- Copyright: AGPLv3

Example:

```
set $tmplist$ = getListContainingList($baseproducts$,$resultlist$)
if stringToBool(compareLists($tmplist$,$baseproducts$))
        comment "check installed products successful finished"
else
        LogError "check installed products failed"
        set $errorList$ = addtolist($errorList$, "check installed products failed")
        ;set $fatal_error$ = "true"
endif
```