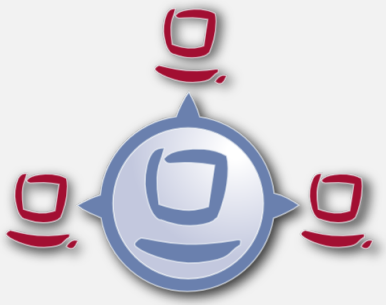

opsi-winst Handbuch (4.11.3)

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

Stand: 11.07.2012

Inhaltsverzeichnis

Kapitel 1

Copyright

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

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

Eine Beschreibung der Lizenz finden Sie hier:
http://creativecommons.org/licenses/by-sa/3.0/de/

Der rechtsverbindliche Text der Lizenz ist hier:
http://creativecommons.org/licenses/by-sa/3.0/de/legalcode

Die Software von opsi ist in weiten Teilen Open Source.
Nicht Open Source sind die Teile des Quellcodes, welche neue Erweiterungen enthalten die noch unter Kofinanzierung stehen, also noch nicht bezahlt sind.
siehe auch:
http://uib.de/www/kofinanziert/index.html

Der restliche Quellcode ist veröffentlicht unter der GPLv3:

Der rechtsverbindliche Text der GPLv3 Lizenz ist hier:
http://www.gnu.org/licenses/gpl.html

Deutsche Übersetzung:
http://www.gnu.de/documents/gpl.de.html

Der Name *opsi* ist eine eingetragene Marke der uib gmbh.

Das opsi-logo ist Eigentum der uib gmbh und darf nur mit ausdrücklicher Erlaubnis verwendet werden.

Kapitel 2

Einführung

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

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

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

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

Kapitel 3

Start und Aufrufparameter

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

Wird der *opsi-winst* ohne Parameter aufgrefufen, so startet er interaktiv.

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

Es existieren folgende Syntax-Varianten des Aufrufs:

(1) Anzeigen der Varianten:

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

(2) Ausführung eines Skripts:

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

(3) Ausführen einer Liste von Skripts:

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

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

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

Generelle Erläuterungen:

- Default Name für die Logdatei ist `C:\tmp\instlog.txt`

- Der Parameterstring, angekündigt durch die Option "/parameter", wird an das jeweils aufgerufene *opsi-winst* Skript (über die String-Funktion ParamStr) übergeben.

Erläuterungen zu (2) und (3):

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

Erläuterungen zu (4):

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

Die Skripte werden per default im Batchmodus abgearbeitet.

3.1 Protokoll Pfade

Die default Protokolldateien werden in das Verzeichnis *c:\tmp* geschrieben, welches der *opsi-winst* zu erstellen versucht. Wenn der *opsi-winst* nicht erfolgreich bei der Erstellung diese Protokollverzeichnisses ist, wird das Benutzer TEMP-Verzeichnis zum Speichern der Protokolldatei genutzt.

Der vorgegebene Name dieser Protokolldatei ist *instlog.txt*. Der Name der Protokolldatei und der Speicherort können durch eine spezifizierte Kommandozeile überschrieben werden.

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

Zusammenfassend ist zu sagen, dass der *opsi-winst* die Protokollverzeichnisse zum Sichern bestimmter temporärer Dateien nutzt.

Kapitel 4

Weitere Konfigurationsoptionen

4.1 Zentrales Protokollieren von Fehlermeldungen

opsi-winst kann die wichtigsten Protokoll-Informationen, insbesondere Fehlermeldungen, auch auf einer zentralen Datei ablegen oder an einen syslog-Dämon schicken.

Dieses Feature lässt sich durch folgende Parameter in der Registry konfigurieren:

In HKEY_LOCAL_MACHINE existiert bei einer Standardinstallation des Programms der Schlüssel \SOFTWARE\opsi.org. In dem Unterschlüssel syslogd wird durch den Wert der Variable remoteerrorlogging festgelegt, ob und wenn ja auf welche Weise ein zentrales Logging stattfindet. In dem Registry-Key

HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\syslogd

sind folgende drei Variable zu betrachten:

- Wenn remoteerrorlogging den Wert 0 hat, findet kein zentrales Logging statt.
- Wenn remoteerrorlogging den Wert 2 hat, werden die Fehlerberichte an einen laufenden Syslog-Dämon geschickt. Der verwendete Hostname wird durch den Inhalt der Variable sysloghost bestimmt (default ist localhost), die Kanalnummer durch den Wert von syslogfacility (default 18, das ist lokal 2).

Die zulässigen Werte für das Facility ergeben sich aus der folgenden Übersicht:

```
ID_SYSLOG_FACILITY_KERNEL     = 0;  // kernel messages
ID_SYSLOG_FACILITY_USER       = 1;  // user-level messages
ID_SYSLOG_FACILITY_MAIL       = 2;  // mail system
ID_SYSLOG_FACILITY_SYS_DAEMON = 3;  // system daemons
ID_SYSLOG_FACILITY_SECURITY1  = 4;  // security/authorization messages (1)
ID_SYSLOG_FACILITY_INTERNAL   = 5;  // messages generated internally by syslogd
ID_SYSLOG_FACILITY_LPR        = 6;  // line printer subsystem
ID_SYSLOG_FACILITY_NNTP       = 7;  // network news subsystem
ID_SYSLOG_FACILITY_UUCP       = 8;  // UUCP subsystem
ID_SYSLOG_FACILITY_CLOCK1     = 9;  // clock daemon (1)
ID_SYSLOG_FACILITY_SECURITY2  = 10; // security/authorization messages (2)
ID_SYSLOG_FACILITY_FTP        = 11; // FTP daemon
ID_SYSLOG_FACILITY_NTP        = 12; // NTP subsystem
ID_SYSLOG_FACILITY_AUDIT      = 13; // log audit
ID_SYSLOG_FACILITY_ALERT      = 14; // log alert
ID_SYSLOG_FACILITY_CLOCK2     = 15; // clock daemon (2)
ID_SYSLOG_FACILITY_LOCAL0     = 16; // local use 0  (local0)
ID_SYSLOG_FACILITY_LOCAL1     = 17; // local use 1  (local1)
ID_SYSLOG_FACILITY_LOCAL2     = 18; // local use 2  (local2)
ID_SYSLOG_FACILITY_LOCAL3     = 19; // local use 3  (local3)
ID_SYSLOG_FACILITY_LOCAL4     = 20; // local use 4  (local4)
ID_SYSLOG_FACILITY_LOCAL5     = 21; // local use 5  (local5)
ID_SYSLOG_FACILITY_LOCAL6     = 22; // local use 6  (local6)
ID_SYSLOG_FACILITY_LOCAL7     = 23; // local use 7  (local7)
```

4.2 Skinnable opsi-winst

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

Kapitel 5

Das *opsi-winst* Skript

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

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

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

5.1 Ein Beispiel

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

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

;Welche Windows-Version?
DefVar $MSVersion$

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


[sub_install_win7]
Files_Kopieren_win7
WinBatch_Setup
```

```
[sub_install_winXP]
Files_Kopieren_XP
WinBatch_SetupXP

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

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

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

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

Wie lassen sich die Sektionen oder Abschnitte dieses Skripts lesen?

5.2 Primäre und sekundäre Unterprogramme des *opsi-winst* Skripts

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

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

Für die *opsi-winst* Skripte ist festgelegt, dass die beiden Sektionen mit den Titeln [Initial] und [Actions] (in dieser Reihenfolge) abgearbeitet werden. Alle anderen Sektionen fungieren als Unterprogramme und können in diesen beiden Sektionen aufgerufen werden. Nur in den Sub-Sektionen können dann wiederum Unterprogramme aufgerufen werden.

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

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

Die primären oder Steuerungssektionen umfassen:

- die **Initial**-Sektion (die zu Beginn des Skripts stehen soll und optional ist),
- die **Actions**-Sektion (die der Initial-Sektion folgen soll und wie diese nur einmal in einem Skript stehen kann) sowie
- die **Sub**-Sektionen (Unterprogramme der Actions-Sektion, die auch deren Syntax und Funktionslogik erben).
- die **ProfileActions**-Sektion die je nach script mode (Machine/Login) unterschiedlich interpretiert wird.

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

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

- Files-Sektionen,
- WinBatch-Sektionen,

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

Im Detail wird Bedeutung und Syntax der unterschiedlichen Sektionstypen in den Abschnitten Kapitel 7 und Kapitel 8 behandelt.

5.3 String-Ausdrücke im *opsi-winst* Skript

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

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

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

(Mehr zu diesem Thema in Kapitel Abschnitt 7.3).

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

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

Kapitel 6

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

6.1 Allgemeines

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

Groß- und Kleinbuchstaben gelten als gleichbedeutend.

Es existieren folgende Arten von Werteträgern:

- Globale Text-Konstanten
  enthalten Werte, die *opsi-winst* automatisch ermittelt und die nicht geändert werden können. Vor der Abarbeitung eines Skripts werden ihre Bezeichnungen im gesamten Skript gegen ihren Wert ausgetauscht. Die Konstante `%ScriptPath%` ist die definierte Pfad-Variable, die den Pfad angibt in dem der *opsi-winst* das Skript findet und ausführt. Dies könnte beispielsweise *p:\product* sein. Man müsste dann
  `"%ScriptPath%"`
  in das Skript schreiben, wenn man den Wert
  p:\product
  bekommen möchte.
  Zu beachten sind die Anführungszeichen um die Konstantenbezeichnung.

- Text-Variable oder String-Variable
  entsprechen den gebräuchlichen Variablen in anderen Programmiersprachen. Die Variablen müssen vor ihrer Verwendung mit `DefVar` deklariert werden. In einer primären Sektion kann einer Variable mehrfach ein Wert zugewiesen werden und mit den Werten in der üblichen Weise gearbeitet werden („Addieren“ von Strings, spezielle String-Funktionen).
  In sekundären Sektionen erscheinen sie dagegen als statische Größen. Ihr jeweils aktueller Wert wird bei der Abarbeitung der Sektion für ihre Bezeichnung eingesetzt (so wie es bei Textkonstanten im ganzen Skript geschieht).

- Variablen für String-Listen
  werden mit `DefStringList` deklariert. Eine String-Listenvariable kann ihren Inhalt, also eine Liste von Strings, auf unterschiedlichste Weisen erhalten. Mit String-Listenfunktionen können die Listen in andere Listen überführt oder als Quelle für Einzelstrings verwendet werden.

Im einzelnen:

6.2 Globale Textkonstanten

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

6.2.1 Verwendung

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

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

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

6.2.2 Beispiel

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

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

Gegenwärtig sind folgende Konstanten definiert:

6.2.3 Systemverzeichnis

6.2.3.1 Basissystemverzeichnis

`%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*

6.2.3.2 Gemeinsames (AllUsers) Verzeichnis

`%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%`

6.2.3.3 Aktuelles (logged in oder usercontext) User Verzeichnis

%AppdataDir% or %CurrentAppdataDir% : *c:\Documents and Settings\%USERNAME%\Application Data*

%CurrentStartmenuDir%

%CurrentDesktopDir%

%CurrentStartupDir%

%CurrentProgramsDir%

%CurrentSendToDir%

%CurrentProfileDir% //since 4.11.2.1

6.2.3.4 /AllNtUserProfiles Verzeichnis Konstanten

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

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

%CurrentProfileDir% // since 4.11.2.1
kann statt %UserProfileDir% verwendet werden um `Files`-Sektionen zu erzeugen die sich genauso auch in *userLogInScripten* verwenden lassen.

6.2.4 *opsi-winst* Pfad und Verzeichnisse

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

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

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

%WinstVersion% : Versionsstring des laufenden Winst.

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

%opsiScriptHelperPath%
Entspricht: %ProgramFiles32Dir%\opsi.org\opsiScriptHelper
Pfad in dem Hilfsprogramme, Libraries und ähnliches zur Scriptausführung installiert sein können.
Seit 4.11.3.2

Beispiel:
Der Code:

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

liefert folgenden Log:

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

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

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

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

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

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

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

Else
EndIf
```

6.2.5 Netzwerk Informationen

`%Host%` : Wert der Umgebungsvariable HOST.

`%PCName%`: Wert der Umgebungsvariable PCNAME oder wenn nicht vorhanden COMPUTERNAME.

`%IPName%` : Der DNS Name eines Computers. Normalerweise ist dieser identisch mit dem netbios-Namen und daher wird `%PCName%` neben dem netbios-Namen in Großbuchstaben geschrieben.

`%Username%` : Name des aktuellen Benutzers.

6.2.6 Service Daten

`%HostID%` : FQDN des Clients.

`%opsiserviceURL%`

`%opsiServer%`

`%opsiserviceUser%` : Die Benutzer ID für die es eine Verbindung zum opsi Service gibt.

`%opsiservicePassword%`

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

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

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

6.3 String- (oder Text-) Variable

6.3.1 Deklaration

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

`DefVar` <variable name>

Beispielsweise

```
DefVar $MsVersion$
```

Erklärung:

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

Empfehlung:

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

6.3.2 Wertzuweisung

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

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

<Value> kann jeder String basierte Ausdruck sein (Beispiele dazu im Abschnitt Abschnitt 7.3).

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

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

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

6.3.3 Verwendung von Variablen in String-Ausdrücken

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

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

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

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

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

6.3.4 Sekundäre und Primäre Sektion im Vergleich

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

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

Zuerst müsste das Verzeichnis $MailLocation$ gesetzt werden:

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

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

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

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

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

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

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

6.4 Variable für String-Listen

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

```
DefStringList SMBMounts
```

String-Listen können z.B. die Ausgabe eines Shell-Programms einfangen und dann in vielfältiger Weise weiterverarbeitet und verwendet werden. Genauere Details dazu findet sich in dem Abschnitt Abschnitt 7.4 zur String-Listenverarbeitung.

Kapitel 7

Syntax und Bedeutung der primären Sektionen eines *opsi-winst* Skripts

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

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

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

7.1 Die primären Sektionen

In einem Skript können vier Arten primärer Sektionen vorkommen:

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

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

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

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

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

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

`ProfileActions` Sektion kann in einem normalen Installations script als sub sektion mit speziellen Syntax Regeln dienen. Existiert diese Section in einem Script das als *userLoginScript* aufgerufen wurde, so ist diese Sektion der Programmstart (statt `Actions`). Siehe Kapitel *User Profile Management* im opsi-manual sowie Abschnitt 7.8

7.2 Parametrisierungsanweisungen für den opsi-winst

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

7.2.1 Beispiel

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

Dies bedeutet, dass

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

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

Der Aufbau der Anweisungszeilen und ihre Bedeutung im Einzelnen:

7.2.2 Festlegung der Protokollierungstiefe

Achtung
Die alte Funktion `LogLevel=` ist ab Winst Version 4.10.3 abgekündigt. Um Rückwärtskompatibilität zu alten Skripten zu gewährleisten wird zu dem hiermit gesetzten Loglevel nochmal 4 hinzuaddiert.

Es gibt zwei ähnliche Varianten, um den Loglevel zu spezifizieren:

`SetLogLevel =` <Zahl>

`SetLogLevel =` <STRINGAUSDRUCK>

SetLogLevel definiert die Tiefe der Protokollierung der Operationen. Im ersten Fall kann die Nummer als Integer Wert oder als String-Ausdruck (vgl. Abschnitt 7.3) angegeben werden. Im zweiten Fall versucht der *opsi-winst* den String-Ausdruck als Nummer auszuwerten.

Es sind zehn Detaillierungsgrade wählbar von 0 bis 9

```
0 = nothing (absolut nichts)
1 = essential ("unverzichtbares")
2 = critical (unerwartet Fehler die zu einem Programmabbruch führen)
3 = error (Fehler, die kein laufendes Programm abbrechen)
4 = warning (das sollte man sich nochmal genauer anschauen)
5 = notice (wichtige Aussagen zu dem Programmverlauf)
6 = info (zusätzlich Informationen)
7 = debug (wichtige debug Meldungen)
8 = debug2 (mehr debug Informationen und Daten)
9 = confidential (Passwörter und andere sicherheitsrelevante Daten)
```

7.2.3 Benötigte *opsi-winst* Version

Die Anweisung

`requiredWinstVersion` <RELATIONSSYMBOL> <ZAHLENSTRING>

z.B.

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

lässt den *opsi-winst* überprüfen, ob die geforderte Versionseigenschaften vorliegt. Wenn nicht erscheint ein Fehlerfenster.

Dieses Feature gibt es erst ab opsi-winst Version 4.3 – bei früheren Versionen führt die noch unbekannte Anweisung einfach zu einer Syntaxfehlermeldung (vgl. auch den folgenden Abschnitt). Daher kann das Statement unabhängig von der aktuell benutzen *opsi-winst* Version benutzt werden so lange die erforderliche Version mindestens 4.3 ist.

7.2.4 Reaktion auf Fehler

Zu unterscheiden sind zwei Sorten von Fehlern, die unterschiedlich behandelt werden müssen:

1. fehlerhafte Anweisungen, die der *opsi-winst* nicht "versteht", d.h. deren Interpretation nicht möglich ist (syntaktischer Fehler),
2. aufgrund von "objektiven" Fehlersituationen scheiternde Anweisungen (Ausführungsfehler).

Normalerweise werden syntaktische Fehler in einem PopUp-Fenster für eine baldige Korrektur angezeigt, Ausführungsfehler werden in einer Log-Datei protokolliert und können später analysiert werden.

Das Verhalten des *opsi-winst* bei einem syntaktischen Fehler wird über die Konfiguration bestimmt.

- `ScriptErrorMessages =` <Wahrheitswert>
  Wenn der Wert true ist (Default), werden Syntaxfehler bzw. Warnhinweise zum Skripts als Message-Fenster auf dem Bildschirm angezeigt. Derartige Fehler werden generell nicht in der Logdatei festgehalten, weil die Protokollierung in der Logdatei die objektiven Probleme einer Installation anzeigen soll.
  Für <Wahrheitswert> kann außer true bzw. false hier zwecks einer suggestiveren Bezeichnung auch on bzw. off eingesetzt werden.
- `FatalOnSyntaxError =` <Wahrheitswert>

  – *true* = (default) Bei einem Syntaxfehler wird das Script abgebrochen und *failed* zurückgeliefert. Dem Server wird die Meldung *Syntax Error* übergeben.

  - *false* = Bei einem Syntaxfehler wird das Script `nicht` abgebrochen.
    Der Syntaxfehler wird in jedem Fall als *Critical* in die Logdatei übernommen.
    In jedem Fall wird der Errorcounter um 1 erhöht.
    Seit 4.11.3.2
    In älteren Versionen wird weder gelogged noch abgebrochen.

Die beiden folgenden Einstellungen steuern die Reaktion auf Fehler bei der Ausführung des Skripts.

- `ExitOnError` = <Wahrheitswert>
  Mit dieser Anweisung wird festgelegt, ob bei Auftreten eines Fehlers die Abarbeitung des Skripts beendet wird. Wenn <Wahrheitswert> *true* oder *yes* oder *on* gesetzt wird, terminiert das Programm, andernfalls werden die Fehler lediglich protokolliert (default).
- `TraceMode` = <Wahrheitswert>
  Wird TraceMode eingeschaltet (Default ist false), wird jeder Eintrag ins Protokoll zusätzlich in einem Message-Fenster auf dem Bildschirm angezeigt und muss mit einem OK-Schalter bestätigt werden.

7.2.5 Vordergrund

- `StayOnTop` = <Wahrheitswert>

Mittels StayOnTop = true (oder = on) kann bestimmt werden, dass im Batchmodus das *opsi-winst* Fenster den Vordergrund des Bildschirms in Beschlag nimmt, sofern kein anderer Task den selben Status beansprucht. Im Dialogmodus hat der Wert der Variable keine Bedeutung.

Achtung
Nach Programmiersystem-Handbuch soll der Wert nicht im laufenden Betrieb geändert werden. Zur Zeit sieht es so aus, als wäre ein einmaliges (Neu-) Setzen des Wertes möglich, ein Rücksetzen auf den vorherigen Wert während des Programmlaufs dann aber nicht mehr.

`StayOnTop` steht per Default auf false damit verhindert wird das Fehlermeldungen eventuell nicht sichtbar sind, weil der *opsi-winst* im Vordergrund läuft.

7.2.6 Fenster Modus

- `NormalizeWinst`
  setzt das *opsi-winst* Fenster auf *normal* Modus.
- `IconizeWinst`
  setzt das *opsi-winst* Fenster auf *minimierten* Modus.
- `RestoreWinst`
  setzt das *opsi-winst* Fenster auf *maximierten* Modus.

7.3 String-Werte, String-Ausdrücke und String-Funktionen

Ein String Ausdruck kann

- ein elementarer String-Wert
- ein verschachtelter String-Wert
- eine String-Variable
- eine Verknüpfung von String-Ausdrücken oder
- ein stringbasierter Funktionsaufruf sein.

7.3.1 Elementare String-Werte

Ein elementarer String-Wert ist jede Zeichenfolge, die von doppelten – " – oder von einfachen – ’ – Anführungszeichen umrahmt ist:

"<Zeichenfolge>"

oder

’<Zeichenfolge>’

Zum Beispiel:

```
DefVar $BeispielString$
Set $BeispielString$ = "mein Text"
```

7.3.2 Strings in Strings („geschachtelte“ String-Werte)

Wenn in der Zeichenfolge Anführungszeichen vorkommen, muss zur Umrahmung die andere Variante des Anführungszeichens verwendet werden.

```
DefVar $Zitat$
Set $Zitat$ = ’er sagte "Ja"’
```

Zeichenfolgen, innerhalb derer möglicherweise bereits Schachtelungen von Anführungszeichen vorkommen, können mit
`EscapeString:` <Abfolge von Buchstaben>
gesetzt werden. Z.B. bedeutet

```
DefVar $MetaZitat$
Set $MetaZitat$ = EscapeString: Set $Zitat$ = ’er sagte "Ja"’
```

dass auf der Variablen $MetaZitat$ am Ende exakt die Folge der Zeichen nach dem Doppelpunkt von EscapeString (inklusive des führenden Leerzeichens) steht, also

```
 Set $Zitat$ = ’er sagte "Ja"’
```

7.3.3 String-Verknüpfung

String-Verknüpfung werden mit dem Pluszeichen ("+") gemacht

<String expression> + <String expression>

Beispiel:

```
DefVar $String1$
DefVar $String2$
DefVar $String3$
DefVar $String4$
Set $String1$ = "Mein Text"
Set $String2$ = "und"
Set $String3$ = "dein Text"
Set $String4$ =  $String1$ + " " + $String2$ + " " + $String3$
```

$String4$ hat dann den Wert "Mein Text und dein Text".

7.3.4 String-Ausdrücke

Eine String-Variable der primären Sektion "beinhaltet" einen String-Wert. Ein String-Ausdruck kann einen elementaren String vertreten. Wie ein String gesetzt und definiert wird findet sich in Abschnitt Abschnitt 6.3.

Die folgenden Abschnitte zeigen die Variationsmöglichkeiten von String-Funktionen.

7.3.5 String-Funktionen zur Ermittlung des Betriebssystemtyps

- `GetOS`
  Die Funktion ermittelt das laufende Betriebssystem. Derzeit liefert sie einen der folgenden Werte: "Windows_16" "Windows_95" (auch bei Windows 98 und ME) "Windows_NT" (auch bei Windows 2000 und XP) "Linux"

- `GetNtVersion`
  (abgekündigt: use GetMSVersionInfo)
  Für ein Betriebssystem mit Windows_NT ist eine Type-Nummer und eine Sub Type-Nummer charakteristisch. GetNtVersion gibt den genauen Sub Type-Namen aus. Mögliche Werte sind
  "NT3"
  "NT4"
  "Win2k" (Windows 5.0)
  "WinXP" (Windows 5.1)
  "Windows Vista" (Windows 6)

Bei höheren (als 6.*) Windows-Versionen werden die Versionsnummern (5.2, ... bzw. 6.0 ..) ausgegeben. Z.B. für Windows Server 2003 R2 Enterprise Edition erhält man
"Win NT 5.2" (Windows Server 2003)
In dem Fall, dass kein NT-Betriebssystem vorliegt, liefert die Funktion als Fehler-Wert:
"Kein OS vom Typ Windows NT"

- `GetMsVersionInfo`
  gibt für Systeme des Windows NT Typs die Information über die Microsoft Version als API aus, z.B. produziert ein Windows XP System das Ergebnis
  "5.1"

Tabelle 7.1: Windows Versionen

GetMsVersionInfo	**Windows Version**
5.0	Windows 2000
5.1	Windows XP (Home, Prof)
5.2	XP 64 Bit, 2003, Home Server, 2003 R2
6.0	Vista, 2008
6.1	Windows 7, 2008 R2

siehe auch `GetMsVersionMap`

- `GetSystemType`
  prüft die Architektur des installierten Betriebssystems. Im Fall eines 64 Bit-Betriebssystems ist der ausgegebene Wert *64 Bit System* oder *x86 System*.

7.3.6 String-Funktionen zur Ermittlung von Umgebungs- und Aufrufparametern

- `EnvVar ( <string>)`
  Die Funktion liefert den aktuellen Wert einer Umgebungsvariablen. Z.B. wird durch EnvVar ("Username") der Name des eingeloggten Users ermittelt.

- `ParamStr` Die Funktion gibt den String aus, der im Aufruf von *opsi-winst* nach dem optionalen Kommandozeilenparameter /parameter folgt. Ist der Kommandozeilenparameter nicht verwendet, liefert ParamStr den leeren String.

- `getLastExitCode`
  Die String-Funktion getLastExitCode gibt den ExitCode des letzten Prozessaufrufs der vorausgehenden *WinBatch / DosBatch / ExecWith* Sektion aus.
  Der Aufruf anderer opsi-winst Befehle (wie z.B. einer *Files* Sektion) verändert den gefundenen ExitCode nicht.
  Bei *DosBatch* und *ExecWith* Sektionen erhalten wir den Exitcode des Interpreters. Daher muss in der Regel der gewünschte Exitcode in der Sektion explizit übergeben werden.

Beispiel:

```
DosInAnIcon_exit1
set $ConstTest$ = "1"
set $CompValue$ = getLastExitCode
if ($ConstTest$ = $CompValue$)
        comment "DosBatch / DosInAnIcon  exitcode passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "DosBatch / DosInAnIcon  exitcode failed"
endif

[DosInAnIcon_exit1]
rem create an errolevel= 1
VERIFY OTHER 2> NUL
echo %ERRORLEVEL%
exit %ERRORLEVEL%
```

- `GetUserSID(`<Windows Username>`)`
  gibt die SID für den übergebenen Benutzer an (möglich mit der Domäne als Vorspann in Form von DOMAIN\USER).
- `GetUsercontext`
  Die Funktion gibt den String aus, der im Aufruf von *opsi-winst* nach dem optionalen Kommandozeilenparameter /usercontext folgt. Ist der Kommandozeilenparameter nicht verwendet, liefert `GetUsercontext` den leeren String.

7.3.7 Werte aus der Windows-Registry lesen und für sie aufbereiten

- `GetRegistryStringValue (`<string>`)`
  versucht den übergebenen String-Wert als einen Ausdruck der Form
  [KEY] X
  zu interpretieren; im Erfolgsfall liest die Funktion den (String-) Wert zum Variablennamen *X* des Schlüssels *KEY* der Registry aus. Wird *X* nicht übergeben so wird der Standardwert des Keys ausgelesen.
  Wenn KEY bzw. die Variable X nicht existieren, wird eine Warnung in das Logfile geschrieben und der Leerstring als Defaultwert zurückgegeben.

Zum Beispiel ist

```
GetRegistryStringValue ("[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
    Winlogon] Shell")
```

üblicherweise "Explorer.exe", das default Windows Shell Programm (aber es könnten auch andere Programme als Shell eingetragen sein.)

Zum Beispiel liefert

```
Set  $CompValue$ = GetRegistryStringValue32 ("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-winst-
    test\test-4.0]")
```

wenn der Key vorher mit dem Standardeintrag *standard entry* erzeugt wurde, den folgenden Logeintrag:

```
Registry started with redirection (32 Bit)
Registry key [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-winst-test\test-4.0]  opened
Key closed
The value of the variable "$CompValue$" is now: "standard entry"
```

- `GetRegistryStringValue32(<string>)` → siehe Kapitel 64 Bit-Unterstützung
- `GetRegistryStringValue64(<string>)` → siehe Kapitel 64 Bit-Unterstützung
- `GetRegistryStringValueSysNative(<string>)` → siehe Kapitel 64 Bit-Unterstützung
- `RegString(<string>)`
  wird vor allem benötigt, um Dateinamen in die Form zu wandeln, in der sie in die Registry eingetragen werden, das heißt, jeder Backslash des Arguments der Funktion wird verdoppelt. Z.B. liefert

```
RegString ("c:\windows\system\")
```

den Wert
"c:\\windows\\system\\"

7.3.8 Werte aus Ini-Dateien lesen

Es gibt – aus historischen Gründen – zwei Funktionen, um Werte aus Ini-Dateien zu lesen.

Als Ini-Datei wird dabei jede in "Sektionen" gegliederte Textdatei der Form

```
[Sektion1]
Varname1=Wert1
Varname2=Wert2
...
[Sektion2]
...
```

bezeichnet.

Die allgemeinste Funktion liest den Key-Wert aus einer Sektion der ini-Datei aus. Jeder Parameter kann als ein willkürlicher String-Ausdruck ausgegeben werden:

- `GetValueFromInifile (<FILE>, <SECTION>, <KEY>, <DEFAULTVALUE>)`
  Die Funktion öffnet die Ini-Datei namens FILE und sucht in deren Sektion SECTION den KEY (auch Variable genannt). Wenn diese Operation erfolgreich sind, wird der zum KEY gehörende Wert zurückgegeben, andernfalls DEFAULTVALUE.
- `GetIni ( <Stringausdruck> [ <character sequence> ] <character sequence> )`
  (abgekündigt: use GetValueFromInifile)
  Diese Funktion unterstützt eine Schreibweise, die sehr eng an die Schreibweise der Ini-Datei selbst angelehnt ist. Dabei kann allerdings nur der Dateiname durch einen String-Ausdruck dargestellt werden, die anderen Größen müssen explizit angegeben sein:
  Der <Stringausdruck> wird als Dateiname ausgelesen, der Erste <character sequence> als Sektionsname, der Zweite als Schlüsselname.
  GetIni("MEINEINIDATEI"[meinesektion] meinkey)
  gibt diese selben Werte zurück wie
  GetValueFromInifile("MEINEINIDATEI","meinesektion","meinkey","")

7.3.9 Produkt Properties auslesen

- `GetProductProperty ( <PropertyName>, <DefaultValue>)`
  wobei <PropertyName> und <DefaultValue> String-Ausdrücke sind. Die Funktion liefert die client-spezifischen Property-Werte für das aktuell installierte Produkt aus.
  Auf diese Weise können PC-spezifische Varianten einer Installation konfiguriert werden.
  Außerhalb des opsi-service Kontextes oder wenn aus anderen Gründen der Aufruf fehlschlägt, wird der angegebene Defaultwert zurückgegeben.

So wurde beispielsweise die opsi UltraVNC Netzwerk Viewer Installation mit folgenden Produkt Properties konfiguriert:

- viewer = <yes> | <no>
- policy = <factory_default> |

Innerhalb des Installationsskript werden die ausgewählten Werte wie folgt abgerufen

```
GetProductProperty("viewer", "yes")
GetProductProperty("policy", "factory_default"
```

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

7.3.10 Informationen aus etc/hosts entnehmen

- `GetHostsName(<string>)`
  liefert den Host-Namen zu einer gegebenen IP-Adresse entsprechend den Angaben in der Host-Datei (die, falls das Betriebssystem laut Environment-Variable OS "Windows_NT" ist, im Verzeichnis "%systemroot%\system32\drivers\etc\" gesucht wird, andernfalls in "C:\Windows\").
- `GetHostsAddr(<string>)`
  liefet die IP-Adresse zu einem gegebenen Host- bzw. Aliasnamen entsprechend der Auflösung in der Host-Datei.

7.3.11 String-Verarbeitung

- `ExtractFilePath(<string>)`
  interpretiert den übergebenen String als Datei- bzw. Pfadnamen und gibt den Pfadanteil (den String-Wert bis einschließlich des letzten “\“ zurück).
- `StringSplit (`STRINGWERT1, STRINGWERT2, INDEX)`
  (abgekündigt: use `splitString` / `takestring`)
- `takeString(<index>,<list>)`
  liefert aus der String-Liste <list> den String mit dem Index <index>.
  Häufig verwendet in Kombination mit splitString : `takeString(<index>, splitString(<string1>, <string2>)`
  (siehe den Abschnitt String-Listenverarbeitung).
  Das Ergebnis ist, dass <string1> in Stücke zerlegt wird, die jeweils durch <string2> begrenzt sind, und das Stück mit <index> (Zählung bei 0 beginnend) genommen wird.

Zum Beispiel ergibt

```
takeString(3, splitString ("\\server\share\directory",  "\"))
```

den Wert *"share"*,
denn mit \ zerlegt sich der vorgegebene String Wert in die Folge:
Index 0 - "" (Leerstring), weil vor dem ersten "\" nichts steht
Index 1 - "" (Leerstring), weil zwischen erstem und zweitem "\" nichts steht
Index 2 - "server"
Index 3 - "share"
Index 4 - "directory"

`takestring` zählt abwärts, wenn der Index negativ ist, beginnend mit der Zahl der Elemente.

Deswegen

```
takestring(-1, $list1$)
```

bedeutet das letzte Element der String-Liste $list1$.

- `SubstringBefore(<string1>, <string2>)`
  (abgekündigt: use splitString / takestring) liefert das Anfangsstück von <string1>, wenn <string2> das Endstück ist. Z.B. hat

```
SubstringBefore ("C:\programme\staroffice\program\soffice.exe", "\program\soffice.exe")
```

den Wert *"C:\programme\staroffice"*.

- `takeFirstStringContaining(<list>,<search string>)`
  Liefert den ersten String einer Liste der <search string> enthält. Liefert einen leeren String, wenn kein passender String gefunden wird.
- `Trim(<string>)`
  Schneidet Leerzeichen am Anfang und Ende des <string> ab.
- `lower(<string>)`
  liefert <string> in Kleinbuchstaben
- `contains(<str>, <substr>)`
  Boolsche Funktion welche *true* liefert wenn <substr> in <str> enthalten ist. Die Funktion arbeitet case sensitive.
  Seit 4.11.3
  Beispiel:

```
set $ConstTest$ = "1xy451Xy451XY45"
set $CompValue$ ="xy"
if contains($ConstTest$, $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $CompValue$ ="xY"
if not(contains($ConstTest$, $CompValue$))
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `stringReplace(<string>, <oldPattern>, <newPattern>)`
  Liefert einen String in dem in <string>, <oldPattern> durch <newPattern> ersetzt ist. Die Funktion arbeitet nicht case sensitive.
  Seit 4.11.3
  Beispiel:

```
set $ConstTest$ = "123451234512345"
set $CompValue$ = stringReplace("1xy451Xy451XY45","xy","23")
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `strLength(<string>)`
  Liefert Anzahl der Zeichen in <string>
  Seit 4.11.3
  Beispiel:

```
set $tmp$ = "123456789"
set $ConstTest$ = "9"
set $CompValue$ = strLength($tmp$)
if $ConstTest$ = $CompValue$
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `strPos(<string>, <sub string>)`
  Liefert die Position des ersten Vorkommens von <sub string> in <string>. Wird <sub string> nicht gefunden so liefert die Funktion "0". Die Funktion arbeitet case sensitive.
  Seit 4.11.3
  Beispiel:

```
set $tmp$ = "1xY451Xy451xy45"
set $ConstTest$ = "7"
set $CompValue$ = strPos($tmp$,"Xy")
if $ConstTest$ = $CompValue$
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $tmp$ = lower("1xY451Xy451xy45")
set $ConstTest$ = "2"
set $CompValue$ = strPos($tmp$,lower("xy"))
if $ConstTest$ = $CompValue$
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `strPart(<string>, <start pos>, <number of chars>)`
  Liefert einen Teilstring von <string> beginnend mit <start pos> und <number of chars> lang. Wenn ab <str pos> weniger als <number of chars> vorhanden sind, so wird der String bis zum Ende geliefert.

Die Zählung der Zeichen beginnt mit 1.
Seit 4.11.3
Beispiel:

```
set $tmp$ = "123456789"
set $ConstTest$ = "34"
set $CompValue$ = strPart($tmp$,"3","2")
if $ConstTest$ = $CompValue$
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
set $tmp$ = "123456789"
set $ConstTest$ = "56789"
set $CompValue$ = strPart($tmp$, strPos($tmp$,"56"),strLength($tmp$))
if $ConstTest$ = $CompValue$
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `unquote(<string>,<quote-string>)`
  Wenn <string> mit dem Anführungszeichen <quote-string> versehen ist so liefert diese Funktion <string> ohne Anführungszeichen
  Von <quote-string> wird nur das erste Zeichen berücksichtigt, wobei führende Whitespaces ignoriert werden.
  Seit 4.11.2.1

```
set $ConstTest$ = "b"
                set $CompValue$ = unquote("'b'", "'")
                comment "compare values"
                if ($ConstTest$ = $CompValue$)
                        comment "passed"
                else
                        set $TestResult$ = "not o.k."
                        LogWarning "failed"
                endif
                comment "double quote"
                set $ConstTest$ = "b"
                set $CompValue$ = unquote('"b"', '"')
                comment "compare values"
                if ($ConstTest$ = $CompValue$)
                        comment "passed"
                else
                        set $TestResult$ = "not o.k."
                        LogWarning "failed"
                endif
                comment "quote string will be trimmed and then only the first char is used"
                comment "note: brackets are different chars"
                set $ConstTest$ = "b]"
                set $CompValue$ = unquote("[b]", " [{ ")
                comment "compare values"
                if ($ConstTest$ = $CompValue$)
                        comment "passed"
```

```
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
comment "not usable to remove brackets"
set $ConstTest$ = "b]"
set $CompValue$ = unquote("[b]", "[")
set $CompValue$ = unquote($CompValue$,"]")
set $CompValue$ = unquote("[b]", "]")
set $CompValue$ = unquote($CompValue$,"[")
set $CompValue$ = unquote(unquote("[b]", "["),"]")
comment "compare values"
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
comment "if string not quoted it will be come back without changes"
set $ConstTest$ = "b"
set $CompValue$ = unquote("b", "'")
comment "compare values"
if ($ConstTest$ = $CompValue$)
        comment "passed"
else
        set $TestResult$ = "not o.k."
        LogWarning "failed"
endif
```

- `HexStrToDecStr(<string>)`
  wandelt einen String mit einem hexadezimalen Zahlwert in einen String mit dem entsprechenden decimalen Wert um. Enthält der Eingabe String führende Zeichen wie *0x* oder *$* werden diese ignoriert.
  Im Fehlerfall: Leerstring

- `DecStrToHexStr(<string>)`+ liefert einen String mit der hexadezimalen Darstellung des Strings zurück, wenn dieser die dezimale Darstellung eines Intergerwertes war. Im Fehlerfall: Leerstring

- `base64EncodeStr(<string>)`
  liefert einen String mit dem base64 encodedten Wert von <string>.

- `base64DecodeStr(<string>)`
  liefert einen String mit dem base64 decodedten Wert von <string>.

7.3.12 Weitere String-Funktionen

- `RandomStr`
  liefert Zufallsstrings (der Länge 10), die aus Klein- und Großbuchstaben sowie Ziffern bestehen. Genauer: 2 Kleinbuchstaben, 2 Großbuchstaben, 2 Sonderzeichen und 4 Ziffern. Die möglichen Sonderzeichen sind:
  !,$,(,),,+,/,;,=,?,[,],{,},ß,~,§,°*

- `CompareDotSeparatedNumbers(<string1>, <string2>)`
  vergleicht zwei Strings vom Typ <zahl>.<zahl>[.<zahl>[.<zahl>]] und liefert "0" bei Gleichheit, "1" wenn string1 größer ist und "-1" wenn string1 kleiner ist.

Beispiel:
Der Code:

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

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

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

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

liefert folgenden Log:

```
comment: Testing:
message CompareDotSeparatedNumbers

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

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

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

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

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

Else
EndIf

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

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

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

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

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

Else
EndIf

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

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

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

Set  $CompValue$ = CompareDotSeparatedNumbers($string1$, $string2$)
```

```
  The value of the variable "$CompValue$" is now: "-1"

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

Else
EndIf
```

- `CompareDotSeparatedStrings` (<string1>, <string2>)
  vergleicht zwei Strings vom Typ <string>.<string>[.<string>[.<string>]] und liefert "0" bei Gleichheit, "1" wenn string1 größer ist und "-1" wenn string1 kleiner ist. Der Vergleich ist nicht Case sensitive.

Beispiel:
Der Code:

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

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

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

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

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

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

liefert folgenden Log:

```
comment: Testing:
message CompareDotSeparatedStrings

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

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

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

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

If
```

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

Else
EndIf

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

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

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

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

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

Else
EndIf

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

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

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

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

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

Else
EndIf

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

Set  $string2$ = "1.a.b.c.3"
```

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

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

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

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

Else
EndIf

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

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

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

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

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

Else
EndIf

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

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

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

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

If
  $ConstTest$ = $CompValue$   <<< result true
  ($ConstTest$ = $CompValue$)   <<< result true
Then
```

```
  comment: passed
  comment: 1.2.3.4.5 is higher then 1.2.13.4.5
  comment: using CompareDotSeparatedStrings give wrong results on numbers

Else
EndIf
```

- `getDiffTimeSec`
  liefert einen String mit dem ganzahligen Wert der vergangenen Sekunden seit dem letzten Aufruf von `marktime`.
  Seit Version 4.11.3.1

- `GetMyIpByTarget(`<target ip addr>`)`
  liefert einen String mit der IP-Adresse des Interfaces, welches das Betriebssystem verwenden wird, wenn es versucht <target ip addr> zu erreichen. Diese Funktion liefert sicherer den korrekten Wert als die Verwendung der Konstante `%IPAddress%`.
  Seit Version 4.11.3.1
  Beispiel:

```
set $CompValue$ = getMyIpByTarget("%opsiServer%")
```

- `GetIpByName(`<ip addr / ip name>`)`
  liefert die IP-Adresse des mit <ip addr / ip name> angegebenen Rechners
  Seit Version 4.11.3.2

```
set $ConstTest$ = "%IPAddress%"
                set $string1$ = "%IPAddress%"
                set $CompValue$ = getIpByName($string1$)
                if ($ConstTest$ = $CompValue$)
                        comment "passed"
                else
                        set $TestResult$ = "not o.k."
                        LogWarning "failed"
                endif
                set $CompValue$ = getIpByName("%HostID%")
                if ($ConstTest$ = $CompValue$)
                        comment "passed"
                else
                        set $TestResult$ = "not o.k."
                        LogWarning "failed"
                endif
                set $CompValue$ = getIpByName("%PCName%")
                if ($ConstTest$ = $CompValue$)
                        comment "passed"
                else
                        set $TestResult$ = "not o.k."
                        LogWarning "failed"
                endif
```

- `SidToName(`<well known sid>`)`
  liefert einen String mit dem lokalisierten Namen der mit <well known sid> bezeichneten Gruppe. Zum Beispiel für *S-1-5-32-544* je nach Lokalisierung des Betriebsystems *Administratoren* oder *Administrators*.
  Seit Version 4.11.3.1

7.3.13 (String-) Funktionen für die Lizenzverwaltung

- `DemandLicenseKey(`poolId [, productId [,windowsSoftwareId]])`
  Über die opsi-Webservicefunktion getAndAssignSoftwareLicenseKey wird vom opsi Service abgefragt, ob es für den Computer eine reservierte Lizenz gibt.

Die Datenbasis auf Grund deren die Lizenzen vergeben werden, kann die Computer ID sein, die Produkt ID oder die Windows Software ID (diese Möglichkeiten bestehen, wenn diese Vorgaben in der Lizenzkonfiguration definiert ist).

poolId, productId, windowsSoftwareId sind Strings (bzw. String-Ausdrücke). Wenn die licensePoolId nicht explizit gesetzt ist, bleibt der erste Parameter ein leerer String "". Das gilt auch für die anderen IDs – sofern diese nicht näher definiert werden.

Die Funktion gibt den Lizenzschlüssel zurück, der aus der Datenbasis ausgewählt wurde.

Beispiele:

```
set $mykey$ = DemandLicenseKey ("pool_office2007")
set $mykey$ = DemandLicenseKey ("", "office2007")
set $mykey$ = DemandLicenseKey ("", "", "{3248F0A8-6813-11D6-A77B}")
```

- `FreeLicense(`poolId [, productId [,windowsSoftwareId]]])`
  Über die Funktion freeSoftwareLicense des opsi Services wird die aktuell belegte Lizenz frei gegeben. Diese Syntax ist analog zum Syntax DemandLicenseKey zu sehen: Beispiel:

```
DefVar $opsiresult$
set $opsiresult$ = FreeLicense("pool_office2007")
```

$opsiresult$ wird zu einem leeren String umgesetzt, wenn kein Fehler auftritt und wenn eine Fehler auftritt, wird der Fehlertext ausgegeben.

7.3.14 Abrufen der Fehlerinformationen von Serviceaufrufen

- `getLastServiceErrorClass`
  liefert einen String zurück, welcher den Namen der Fehlerklasse des letzten Serviceaufrufs zurück. Wenn der letzte Serviceaufruf keine Fehlermeldung verursacht hat, gibt die Funktion den Wert "None" zurück.
- `getLastServiceErrorMessage`
  liefert einen String zurück, welcher die Fehlermeldung des letzten Serviceaufrufs entspricht. Wenn der letzte Serviceaufruf keine Fehlermeldung verursacht hat, gibt die Funktion den Wert "None" zurück.

Da die Nachrichtenstrings sich immer mal wieder ändern, wird für die Logik des Grundskriptes die Verwendung des Klassennamen empfohlen.

Beispiel:

```
if getLastServiceErrorClass = "None"
    comment "kein Fehler aufgetreten"
endif
```

7.4 String-Listenverarbeitung

Eine String-Liste (oder ein String-Listenwert) ist ein Sequenz eines String-Wertes. Für diese Werte gibt es die Variable der String-Listen. Sie sind wie folgt definiert

`DefStringList` <VarName>

Ein String-Listenwert ist einer String-Listenvariable zugeteilt:

`Set` <VarName> `=` <StringListValue>

String-Listen können auf vielfältige Weise erzeugt bzw. „eingefangen“ werden. Sie werden in String-Listen-Ausdrücken verarbeitet. Der einfachste String-Listen-Ausdruck ist das Setzen eines (String-Listen-) Wertes auf eine (String-Listen-) Variable.

Für die folgenden Beispiele sei generell eine String-Listen-Variable *$list1$* definiert:

```
DefStringList $list1$
```

Diese Variable lässt sich auf ganz unterschiedliche Weise mit Inhalten füllen: Wenn wir Variablen mit <String0>, <StringVal>, .. benennen bedeutet das, dass diese für jeden belieben String-Ausdruck stehen können.

Wir beginnen mit einer speziellen und sehr hilfreichen Art von String-Listen: Funktionen – also aufgerufene Hashes oder zugehörige Arrays – welche aus einer Zeile von dem Aufruf *KEY=VALUE* stammen. Tatsache ist, dass jede Funktion eine Funktion ermitteln sollte, welche einen VALUE mit einem KEY assoziiert. Jeder KEY sollte in dem ersten Abschnitt einer Zeile auftreten (während verschiedene KEYs mit identischen VALUE verbunden sein können).

7.4.1 Info Maps

- `getMSVersionMap`
  fragt die Betriebssysteminformationen lokal ab und schreibt die Informationen in eine String-Liste.

Im Moment existieren folgende Schlüssel

- major_version
- minor_version
- build_number
- platform_id
- csd_version
- service_pack_major
- service_pack_minor
- suite_mask
- product_type_nr
- 2003r2

Die Ergebnisse von suite_mask und product_type_nr sind Zahlen, die aus bitweisen-or-Verknüpfungen der folgenden Werte gebildet sein können.

product_type_nr

```
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)
```

Beispiel:
Der Code

```
DefStringList $INST_Resultlist$
DefStringList $INST_Resultlist2$

message "getMSVersionMap"
comment "get value by winst function"
set $INST_Resultlist$ = getMSVersionMap
```

Liefert z.B. im Log:

```
message getMSVersionMap
comment: get value by winst function

Set  $INST_Resultlist$ = getMSVersionMap
    retrieving strings from getMSVersionMap [switch to loglevel 7 for debugging]
        (string   0)major_version=5
        (string   1)minor_version=1
        (string   2)build_number=2600
        (string   3)platform_id=2
        (string   4)csd_version=Service Pack 3
        (string   5)service_pack_major=3
        (string   6)service_pack_minor=0
        (string   7)suite_mask=256
        (string   8)product_type_nr=1
        (string   9)2003r2=false
```

Anmerkung
Background infos for getMSVersionMap

- http://msdn.microsoft.com/en-us/library/ms724385%28VS.85%29.aspx
- http://msdn.microsoft.com/en-us/library/dd419805.aspx
- http://msdn.microsoft.com/en-us/library/ms724833%28VS.85%29.aspx

- `getFileInfoMap(<FILENAME>)`
  findet die Versionsinformationen, die im FILENAME verborgen sind und schreibt sie in eine String-Listen Funktion.

Zur Zeit existieren folgende Schlüssel,

- Comments

- CompanyName
- FileDescription
- FileVersion
- InternalName
- LegalCopyright
- LegalTrademarks
- OriginalFilename
- PrivateBuild
- ProductName
- ProductVersion
- SpecialBuild
- Language name <index>
- Language ID <index>
- file version with dots
- file version
- product version

Verwendung: Wenn wir folgende definieren und aufrufen

```
DefStringList FileInfo
DefVar $InterestingFile$
Set $InterestingFile$ = "c:\program files\my program.exe"
set  FileInfo = getFileInfoMap($InterestingFile$)
```

bekommen wir die Werte, die zum Schlüssel "FileVersion" dazugehören, über den Aufruf

```
DefVar $result$
set $result$ = getValue("FileVersion", FileInfo)
```

ausgegeben (für die Funktion getValue vgl. Abschnitt Abschnitt 7.4.4).

Beispiel:
Der Code:

```
set $InterestingFile$ = "%winstdir%\winst.exe"
if not (FileExists($InterestingFile$))
        set $InterestingFile$ = "%winstdir%\winst32.exe"
endif
set $INST_Resultlist$ = getFileInfoMap($InterestingFile$)
```

liefert z.B. im Log

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

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

```
Then
EndIf

Set  $INST_Resultlist$ = getFileInfoMap($InterestingFile$)
    retrieving strings from getFileInfoMap [switch to loglevel 7 for debugging]
        (string   0)Language name 0=Deutsch (Deutschland)
        (string   1)Language ID 0=1031
        (string   2)file version=1125942857039872
        (string   3)file version with dots=4.10.8.0
        (string   4)product version=1125942857039872
        (string   5)Comments=
        (string   6)CompanyName=uib gmbh (www.uib.de)
        (string   7)FileDescription=opsi.org
        (string   8)FileVersion=4.10.8.0
        (string   9)InternalName=
        (string  10)LegalCopyright=uib gmbh under GPL
        (string  11)LegalTrademarks=opsi
        (string  12)OriginalFilename=
        (string  13)PrivateBuild=
        (string  14)ProductName=opsi-winst
        (string  15)ProductVersion=4.0
        (string  16)SpecialBuild=
```

- `getLocaleInfoMap`
  fragt die Systeminformationen lokal ab und schreibt die Informationen in eine String-Liste.

Im Moment existieren folgende Schlüssel

- language_id_2chars (eine „Zwei-Buchstaben“ Namensangabe der default Systemsprache)
- language_id (eine „Drei-Buchstaben“ Namensangabe der default Systemsprache inklusive der Sprachenuntertypen)
- localized_name_of_language
- English_name_of_language
- abbreviated_language_name
- native_name_of_language
- country_code
- localized_name_of_country
- English_name_of_country
- abbreviated_country_name
- native_name_of_country
- default_language_id
- default_language_id_decimal
- default_country_code
- default_oem_code_page
- default_ansi_code_page
- default_mac_code_page

- system_default_language_id Hexadecimal Windows locale Id
- system_default_posix Sprache_Region (Posix Style)
- system_default_lang_region Sprache-Region (BCP 47 Style)

Die system_default Keys geben Informationen über die Sprache des installierten Betriebssystems. Die anderen Keys geben Informationen über die Lokalisierung der GUI.

Beispiel:
Der Code:

```
message "Locale Infos"
set  $INST_Resultlist$ = getLocaleInfoMap
```

liefert z.B. folgendes Log:

```
message Locale Infos

Set  $INST_Resultlist$ = getLocaleInfoMap
    retrieving strings from getLocaleInfoMap [switch to loglevel 7 for debugging]
        (string   0)language_id_2chars=DE
        (string   1)language_id=DEU
        (string   2)localized_name_of_language=Deutsch (Deutschland)
        (string   3)English_name_of_language=German
        (string   4)abbreviated_language_name=DEU
        (string   5)native_name_of_language=Deutsch
        (string   6)country_code=49
        (string   7)localized_name_of_country=Deutschland
        (string   8)English_name_of_country=Germany
        (string   9)abbreviated_country_name=DEU
        (string  10)native_name_of_country=Deutschland
        (string  11)default_language_id=0407
        (string  12)default_language_id_decimal=1031
        (string  13)default_country_code=49
        (string  14)default_oem_code_page=850
        (string  15)default_ansi_code_page=1252
        (string  16)default_mac_code_page=10000
        (string  17)system_default_language_id=0407
        (string  18)system_default_posix=de_DE
        (string  19)system_default_lang_region=de-DE
```

Verwendung: Wenn wir den Aufruf wie folgt definieren

```
DefStringList $languageInfo$
set  $languageInfo$ = getLocaleInfoMap
```

bekommen wir den Wert mit dem KEY "language_id_2chars" über den Aufruf

```
DefVar $result$
set $result$ = getValue("language_id_2chars", $languageInfo$)
```

(für die Funktion getValue vgl. Abschnitt Abschnitt 7.4.4). Wir können nun Skripte mit folgender Konstruktion verwenden

```
if getValue("language_id_2chars", languageInfo) = "DE"
    ; installiere deutsche Version
else
   if getValue("language_id_2chars", languageInfo) = "EN"
```

```
    ; installiere englische Version
   endif
endif
```

Anmerkung
Background infos for getLocaleInfoMap:

- http://msdn.microsoft.com/en-us/library/cc233968.aspx
- http://msdn.microsoft.com/en-us/library/0h88fahh.aspx
- bcp 47 validator:
  http://schneegans.de/lv/?tags=de-de-1996&format=text
- http://www.iana.org/assignments/language-subtag-registry
- http://www.the-localization-tool.com/?p=698

Die Funktion GetLocaleInfoMap ersetzt die ältere GetLocaleInfo, da diese Werte ausliest, die schwierig zu interpretieren sind:

- `getLocaleInfo`
  (abgekündigt): Bitte `GetLocaleInfoMap` verwenden.
- `getProductMap` // since 4.11.2.1
  liefert eine info map über das opsi product welches gerade installiert wird.
  Di Funktion arbeitet nur korrekt, wenn *opsi-winst* im opsi service mode aufgerufen wird.
  keys sind: id, name, description, advice, productversion, packageversion, priority, installationstate, lastactionrequest, lastactionresult, installedversion, installedpackage, installedmodificationtime

Beispiel:

```
set $INST_Resultlist$ = getProductMap
set $string1$ = getValue("id", $INST_Resultlist$)
```

liefert z.B. folgenden log:

```
Set  $INST_Resultlist$ = getProductMap
    retrieving strings from getProductMap [switch to loglevel 7 for debugging]
        (string   0)id=opsi-winst-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=


Set  $string1$ = getValue("id", $INST_Resultlist$)
    retrieving strings from $INST_Resultlist$ [switch to loglevel 7 for debugging]
```

```
        (string   0)id=opsi-winst-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=

  The value of the variable "$string1$" is now: "opsi-winst-test"
```

7.4.2 Erzeugung von String-Listen aus vorgegebenen String-Werten

- `createStringList (`Stringwert0, Stringwert1 ,... `)`
  erzeugt eine neue Liste aus den aufgeführten einzelnen String-Werten, z.B. liefert

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

die ersten vier Buchstaben des Alphabets.

- `splitString (`Stringwert1, Stringwert2)`
  erzeugt die Liste der Teilstrings von String-Wert1, die jeweils durch String-Wert2 voneinander getrennt sind. Z.B. bildet

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

die Liste
"", "", "server", "share", "directory"

- `splitStringOnWhiteSpace (`<string>)
  zerlegt <string> in die durch "leere" Zwischenräume definierten Abschnitte. Das heißt z.B.

```
set $list1$ = splitStringOnWhiteSpace("Status   Lokal     Remote         Netzwerk")
```

liefert die Liste
"Status", "Lokal", "Remote", "Netzwerk"
unabhängig davon, wie viele Leerzeichen oder ggf. Tabulatorzeichen zwischen den "Wörtern" stehen.

7.4.3 Laden der Zeilen einer Textdatei in eine String-Liste

- `loadTextFile (`Dateiname)`
  liest die Zeilen der Datei des (als String) angegebenen Namens ein und generiert aus ihnen eine String-Liste.
- `loadUnicodeTextFile (`Dateiname)`
  liest die Zeilen der Unicode-Datei des (als String) angegebenen Namens ein und generiert aus ihnen eine String-Liste.
  Die Strings in der String-Liste werden dabei (gemäß den Betriebssystem-Defaults) in 8 Bit-Code konvertiert.
- `getSectionNames (`Dateiname)`
  liest die ausgewählte Datei als eine ini-Datei aus, durchsucht alle Zeilen nach dem Begriff *[<SectionName>]* und gibt die einfachen Sektionsnamen (ohne Klammern) aus.

7.4.4 (Wieder-) Gewinnen von Einzelstrings aus String-Listen

- `composeString (`<StringListe>, <LinkString>`)`
  Mit dieser Funktion lässt sich die Zerlegung eines Strings in einer String-Liste z.B. nach vorheriger Transformation (s. den Abschnitt „Transformation von String-Listen“) – rückgängig machen.
  Zum Beispiel:
  Wenn $list1$ für die Liste *a*, *b*, *c*, *d*, *e* steht, erhält die String-Variable line mittels `$line$ = composeString ($list1$, " | ")` den Wert *"a / b / c / d / e"*.

- `takeString(`<index>,<list>`)`
  liefert aus der String-Liste <list> den String mit dem Index <index>
  zum Beispiel liefert (wenn $list1$ wie eben die Liste der ersten fünf Buchstaben des Alphabets ist) `takeString (2, list1)`
  den String *c* (der Index beruht auf einer mit 0 beginnenden Nummerierung der Listenelemente).
  Negative Werte des index werden die Werte abwärts der Liste ausgelesen. Z.B., `takeString (-1, list1)`
  gibt das letzte Listenelement zurück; das ist *e*.

- `takeFirstStringContaining(`<list>,<search string>`)`
  Liefert den ersten String einer Liste der <search string> enthält. Liefert einen leeren String wenn kein passender String gefunden wird.

- `getValue (`<key>, <list>`)`
  Diese Funktion versucht eine String-Liste <list> als eine Liste aus Zeilen der Form
  key=value
  auszulesen.
  Dazu sucht die Funktion die erste Zeile, wo der String-Key vor dem Gleicheitszeichen erscheint und gibt den Rest der Zeile zurück (der String, der nach dem Gleicheitszeichen folgt). Wenn es keinen passende Zeile gibt, wird der Wert *NULL* zurückgegeben.
  Die Funktion ist z.B. notwendig für die Nutzung von `getLocaleInfoMap` und `getFileVersionMap` String list Funktionen (vgl. Abschnitt Abschnitt 7.4.1).

- `getValueBySeparator(`<key string>,<separator string>,<hash string list> `)` //since 4.11.2.1
  arbeitet wie `getValue` aber der Trenner zwischen *key* und *value* (<separator string>) muss angegeben werden so das mit maps gearbeitet werden kann wie
  key:value

Die pseudo-integer Funktion * `count (`<list>`)`
ist eine pseudo Integer Funktion. Sie zählt die Elemente einer String-Liste <list>; das Resultat wird in einen String gewandelt. Ist $list1$ z.B.
a, *b*, *c*, *d*, *e*
so hat `count ($list1$)` den Wert "5".

7.4.5 String-Listen-Erzeugung mit Hilfe von Sektionsaufrufen

- `retrieveSection (`` `Sektionsname`)``
  gibt die Zeilen einer aufgerufene Sektion aus.

- `getOutStreamFromSection (`` `Sectionname`)``
  „fängt“ – derzeit bei `DosInAnIcon` (`ShellInAnIcon`),`ExecWith` und `ExecPython` Aufrufen – die Ausgabe der Kommandozeilenprogramme in der Form einer String-Liste ein. Z.B. liefert der Ausdruck `getOutStreamFromSection ('DosInAnIcon_netuse')`
  wenn die aufgerufene Sektion definiert ist durch

```
[DosInAnIcon_netuse]
net use
```

eine Reihe von Zeilen, die u.a. die Auflistung aller auf dem PC verfügbaren Shares enthalten und dann weiterbearbeitet werden können.

- `getReturnListFromSection (`Sectionname)`
  In Sektionen bestimmter Typen – derzeit implementiert nur für `XMLPatch`-und `opsiServiceCall`-Sektionen – existiert eine spezifische Return-Anweisung, die ein Ergebnis der Sektion als String-Liste zur Verfügung stellt.

`XMLPatch` Beispiel:
Die Anweisung
`set $list1$ =getReturnListFromSection ('XMLPatch_mime "c:\mimetypes.rdf"')`
liefert eine spezifisch selektierte Liste von Knoten der XML-Datei mimetypes.rdf liefern. Näheres zu `XMLPatch`-Sektionen ist der Dokumentation im Kapitel Abschnitt 8.7 zu entnehmen.

`OpsiServiceCall` Beispiel:

```
DefStringList $result$
Set $result$=getReturnListFromSection("opsiservicecall_clientIdsList")

[opsiservicecall_clientIdsList]
"method":"getClientIds_list"
"params":[]
```

7.4.6 String-Listen aus der Registry

- `getRegistryKeyList32(`<regkey>`)`
  Liefert eine Liste mit dem Namen aller Keys direkt unterhalb von <regkey>.
  32 Bit Modus (mit Redirection). Seit 4.11.3
- `getRegistryKeyList64(`<regkey>`)`
  Liefert eine Liste mit dem Namen aller Keys direkt unterhalb von <regkey>.
  64 Bit Modus (ohne Redirection). Seit 4.11.3
- `getRegistryKeyListSysnative(`<regkey>`)`
  Liefert eine Liste mit dem Namen aller Keys direkt unterhalb von <regkey>.
  Modus abhängig von der Architektur des Betriebssystems. Seit 4.11.3
- `getRegistryVarList32(`<regkey>`)`
  Liefert eine Liste mit dem Namen aller Werte direkt unterhalb von <regkey>.
  32 Bit Modus (mit Redirection). Seit 4.11.3
- `getRegistryVarList64(`<regkey>`)`
  Liefert eine Liste mit dem Namen aller Werte direkt unterhalb von <regkey>.
  64 Bit Modus (ohne Redirection). Seit 4.11.3
- `getRegistryVarListSysnative(`<regkey>`)`
  Liefert eine Liste mit dem Namen aller Werte direkt unterhalb von <regkey>.
  Modus abhängig von der Architektur des Betriebssystems. Seit 4.11.3
- `getRegistryVarMap32(`<regkey>`)`
  Liefert eine Map mit den Namen=Value Paaren aller Werte direkt unterhalb von <regkey>.
  32 Bit Modus (mit Redirection). Seit 4.11.3
- `getRegistryVarMap64(`<regkey>`)`
  Liefert eine Map mit den Namen=Value Paaren aller Werte direkt unterhalb von <regkey>.
  64 Bit Modus (ohne Redirection). Seit 4.11.3
- `getRegistryVarMapSysnative(`<regkey>`)`
  Liefert eine Map mit den Namen=Value Paaren aller Werte direkt unterhalb von <regkey>.
  Modus abhängig von der Architektur des Betriebssystems. Seit 4.11.3

Beispiel:
Wir erzeugen Registryeinträge mit folgender Sektion durch den Aufruf von:

```
Registry_createkeys /32Bit

openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-winst-test]
set "var1" = "value1"
set "var2" = REG_SZ:"value2"
set "var3" = REG_EXPAND_SZ:"value3"
set "var4" = REG_DWORD:444
set "var5" = REG_BINARY:05 05 05 0F 10
set "var6" = REG_MULTI_SZ:"value6|value7|de"
openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-winst-test\key1]
openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-winst-test\key2]
openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-winst-test\key3]
openkey [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-winst-test\key4]
```

Dann liefert uns:

```
set $list$ = getRegistryVarList32("hklm\software\opsi.org\opsi-winst-test")
```

folgenden Log:

```
Set   $list$ = GetRegistryVarList32("hklm\software\opsi.org\opsi-winst-test")
Registry started with redirection (32 Bit)
    retrieving strings from GetRegistryVarList32 [switch to loglevel 7 for debugging]
        (string   0)var1
        (string   1)var2
        (string   2)var3
        (string   3)var4
        (string   4)var5
        (string   5)var6
```

Dann liefert uns:

```
set $list$ = getRegistryVarMap32("hklm\software\opsi.org\opsi-winst-test")
```

folgenden Log:

```
Set   $list$ = GetRegistryVarMap32("hklm\software\opsi.org\opsi-winst-test")
retrieving strings from GetRegistryVarMap32 [switch to loglevel 7 for debugging]
    (string   0)var1=value1
    (string   1)var2=value2
    (string   2)var3=value3
    (string   3)var4=444
    (string   4)var5=05 05 05 0F 10
    (string   5)var6=value6
```

Dann liefert uns:

```
set $list$ = getRegistryKeyList32("hklm\software\opsi.org\opsi-winst-test")
```

folgenden Log:

```
Set   $list$ = GetRegistryKeyList32("hklm\software\opsi.org\opsi-winst-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
```

7.4.7 String-Listen aus Produkt Properties

- `getProductPropertyList(<propname>,<default value>)`
  Liefert eine Liste mit den aktiven Werten des multivalue Properties <propname>. Im Gegensatz zu der Funktion `GetProductProperty` welche in diesem Fall die einzelnen Werte auf einem komma separierten String zurück liefert. Diese Vorgehen wird problematisch wenn Kommas auch in der Werten vorkommen.
  Seit 4.11.3
  Beispiel:

```
;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
```

7.4.8 Sonstige String-Listen

- `getProfilesDirList`
  Liefert eine Liste der Pfade zu den lokalen Profilen.
  Profile welche die folgenden Worte enthalten werden **nicht** berücksichtigt:

- *localservice*
- *networkservice*
- *systemprofile*

Das Profil des *Default Users* ist Bestandteil der Liste.
`All User` oder `Public` sind nicht Bestandteil der Liste.

Beispiel:

```
set $list1$ = getProfilesDirList
```

ergibt folgenden Log:

```
Set  $list1$ = getProfilesDirList
Registry started with redirection (32 Bit)
    retrieving strings from getProfilesDirList [switch to loglevel 7 for debugging]
        (string   0)C:\Users\Administrator
        (string   1)C:\Users\Default
```

7.4.9 Transformation von String-Listen

- `getSubList` (<Startindex>, <Endindex>, <list>)
  Liefert eine Teilliste einer vorgegebenen Liste.
  Funktion:
  Wenn $list$ z.B. für die Liste der Buchstaben *a*, *b*, *c*, *d*, *e* steht, so liefert

```
set $list1$ = getSubList(1 : 3, $list$)
```

b, *c*, *d* (Startindex und Endindex sind die Nummer des Listenelements, wenn mit 0 beginnend gezählt wird).
Defaultwert des Startindex ist 0, des Endindex der letzte Index der Liste. Z.B. ergibt mit obiger Festlegung für $list$

```
set $list1$ = getSubList(1 : , $list$)
```

b, *c*, *d*, *e*.

```
set $list1$ = getSubList(:, $list$)
```

ist genau eine Kopie der ursprünglichen Liste.
Es besteht die Möglichkeit den Endindex mit Rückwärtszählung zu bestimmen:

```
set $list1$ = getSubList(1 : -1, $list$)
```

ist die Teilliste der Elemente vom 1. bis zum vorletzten Element der ursprünglichen Liste – im obigen Beispiel also wieder *b*, *c*, *d*.

- `getListContaining`(<list>,<search string>)
  Liefert eine Teilliste mit allen Strings welche den <search string> enthalten.
- `takeFirstStringContaining`(<list>,<search string>)
  Liefert den ersten String von <list> welcher den <search string> enthält. + Liefert einen Leerstring wenn <search string> nicht gefunden wird.
- `addtolist`(<list>,<string>)
  Hängt den String <string> an die Liste <list> an.
- `addlisttolist`(<list1>,<list2>)
  Hängt die Liste <list2> an die Liste <list1> an.
- `reverse` (<list>)
  kehrt die Reihenfolge der Aufzählung um – aus *a*, *b*, *c*, *d*, *e* wird mit

```
set $list1$ = reverse ($list$)
```

also *e*, *d*, *c*, *b*, *a*.

7.4.10 Iteration durch String-Listen

Eine besonders wichtige Anwendung von String-Listen beruht auf der Möglichkeit, die Elemente einer String-Liste zu durchlaufen und für jedes Element eine vorgegebenes Anweisungsschema auszuführen:

Die Syntax für eine solche Iteration („Wiederholungsanweisung“) lautet:

- `for` %s% `in` <list> `do` <eine Anweisung>

Dabei wird %s% durch diesen Ausdruck und nur für diese Stelle als String-Variable deklariert und ist danach wieder unbekannt. Innerhalb der Anweisung wird jedes Vorkommen von %s% (oder wie auch immer eine entsprechende Variable benannt ist) der Reihe nach durch die verschiedenen Elemente der Liste ersetzt.

Achtung
Die Ersetzung ist (wie bei Systemkonstanten) rein textuell, d.h. genau die Zeichenfolge %s% wird z.B. durch die Werte a b c - ersetzt. Sind die Strings *a,b,c* gemeint, muss in der auszuführenden Anweisung %s% von Anführungszeichen eingeschlossen sein.

Ein Beispiel: Wenn $list1$ für *a*, *b*, *c*, *d*, *e* steht und $line$ als String-Variable deklariert ist, so bedeutet:

```
for %s% in $list1$ do   set $line$ = $line$ + "%s%"
```

der Reihe nach

```
$line$ = $line$ + "a"
$line$ = $line$ + "b"
$line$ = $line$ + "c"
$line$ = $line$ + "d"
$line$ = $line$ + "e"
```

so dass am Ende $line$ den Wert *abcde* trägt. Wenn wir die einfachen Anführungszeichen um das %s% weglassen würden, bekämen wir bei jedem Schritt der Iteration einen Syntaxfehler gemeldet.

7.5 Spezielle Kommandos

- `Killtask` <process> stoppt alle Prozesse, in denen das durch <process> bezeichnete Programm ausgeführt wird. Beispiel :

```
killtask "winword.exe"
```

- `ChangeDirectory` <directory> //since 4.11.2.6 Setzt das angegebene Directory als Arbeitsverzeichnis des *opsi-winst*. Wirkt auf alle nachfolgenden Aktionen (z.B. winbatch Sektionen) und wird am Ende einese Scriptes automatisch zurückgesetzt. Beispiel :

```
ChangeDirectory "%SCRIPTPATH%\programm"
```

- `sleepSeconds` <Integer>
  unterbricht die Programmausführung für <Integer> Sekunden.
- `markTime`
  Setzt einen Zeitstempel für die Systemlaufzeit und zeichnet diese auf.
- `diffTime`
  Zeichnet die vergangene Zeit seit der letzten aufgezeichneten Zeit (`marktime`) auf.

7.6 Kommandos zur Steuerung des Logging

- `comment` <STRINGAUSDRUCK>
  bzw.
  `comment` = <Zeichensequenz>
  wird einfach der Wert des String-Ausdrucks (bzw. Zeichensequenz) als Kommentar in die Protokolldatei eingefügt.
- `LogError` <STRINGAUSDRUCK>
  oder
  `LogError` = <Zeichensequenz>
  Fügt eine zusätzlich Fehlermeldungen in der Protokolldatei ein und erhöht den Fehlerzähler um eins.
- `LogWarning` <STRINGAUSDRUCK>
  oder
  `LogWarning` = <Zeichensequenz>
  Fügt eine zusätzlich Warnmeldungen in der Protokolldatei ein und erhöht den Warnungszähler um eins.
- `includelog` <file name> <tail size> //since 4.11.2.1
  Fügt die Datei <file name> in den aktuellen ein. Dabei werden nur die letzten <tail size> Zeilen und nicht die komplette Logdatei eingefügt. Wenn Sie ein anderes Programm (z.B. ein setup programm) starten das eine Logdatei produziert, können Sie mit diesem Begfehl die Informationen aus dieser Logdatei in den Log des *opsi-winst* übernehmen.
  Seit Version 4.11.3.2 kann auch eine negative <tail size> angegeben werden. Dann arbeitet `includelog` im *Head* Modus, d.h. ist <tail size> = -5, so werden die ersten 5 Zeilen von <file name> in den Log übernommen. Beispiel:

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

7.7 Anweisungen für Information und Interaktion

- `Message` <STRINGAUSDRUCK>
  bzw.
  `Message` = <Buchstabenfolge>
  bewirkt, dass in der Batch-Oberfläche des *opsi-winst* der Wert von STRINGAUSDRUCK bzw. dass die Buchstabenfolge als Hinweis-Zeile zum gerade laufenden Installationsvorgang angezeigt wird (solange bis die Installation beendet ist oder eine andere `message` angefordert wird).
  Empfohlen ist die erste Variante, da nur hier auch Variablen für den Messagetext verwendet werden können. Beispiel:

```
Message "Installation von "+$productid$
```

- `ShowMessageFile` <file name>
  zeigt den Inhalt der Datei <file name> in einem gesonderten Fenster an und wartet auf Bestätigung durch den Anwender. Z.B. könnte so eine "Nachricht des Tages" angezeigt werden:

```
ShowMessageFile "p:\login\day.msg"
```

- `ShowBitMap` [<DATEINAME>] [<Beschriftung>]
  wird die Bilddatei <DATEINAME> (BMP-, JPEG- oder PNG-Format, 160x160 Pixel) in der Batch-Oberfläche angezeigt. Eine Beschriftung kann ebenfalls hinzugefügt werden. <DATEINAME> und <Beschriftung> sind String-Ausdrücke. Wenn der Namensparameter fehlt, wird das Bild gelöscht. Beispiel:

```
ShowBitmap "%scriptpath%\" + $ProduktName$ + ".bmp"  "$ProduktName$"
```

- `Pause` <STRINGAUSDRUCK>
  bzw.
  `Pause` = <Zeichensequenz>
  Die beiden Anweisungen zeigen in einem Textfenster den in STRINGAUSDRUCK gegebenen Text (bzw. Zeichensequenz) an. Auf Anklicken eines Knopfes setzt sich der Programmlauf fort.
- `Stop` <STRINGAUSDRUCK>
  bzw.
  `stop` = <Zeichensequenz>
  der STRINGAUSDRUCK (bzw. Zeichensequenz, die möglicherweise auch leer ist) wird angezeigt und um Bestätigung gebeten, dass der Programmablauf abgebrochen werden soll.

7.8 Commands for userLoginScripts / User Profile Management

Anmerkung
Die opsi Erweiterung *User Profile Management* ist derzeit (19.10.2011) ein Kofinanzierungsprojekt. Dies bedeutet, dass diese Erweiterung nicht frei ist, sondern käuflich erworben werden muss um sie zu verwenden.

- `GetScriptMode` //since 4.11.2.1
  liefert eines der beiden Werte *Machine*,*Login*:
  - *Machine* - das Script läuft **nicht** als *userLoginScript*
  - *Login* - das Script läuft als *userLoginScript*
- `GetUserSID(`<Windows Username>`)`
- `GetLoggedInUser` //since 4.11.1.2
- `GetUsercontext` //since 4.11.1.2
  liefert den Namen des Users in dessen Kontext der *opsi-winst* gerade läuft.
- `saveVersionToProfile` //since 4.11.2.1
  speichert `productversion-packageversion` die im lokalen Profil
  Diese Funktion ist gedacht für *userLoginScripts*.
  Diese Funktion kann in Kombination mit `readVersionFromProfile` verwendet werden um festzustellen ob ein Script schonmal gelaufen ist. Es speichert im Lokalen Profil (in der Datei "%CurrentAppdata-Dir%\.opsi.org\userLoginScripts.ini"), dass das *userLoginScript* für dieses opsi product in dieser product version und package version für den aktuellen user ausgefürt wurde. Sieh auch `scriptWasExecutedBefore`
- `readVersionFromProfile` //since 4.11.2.1
  liefert einen string mit `productversion-packageversion` für das aktuelle opsi produkt der aus dem lokalen Profil ausgelesen wird. Siehe auch: `saveVersionToProfile`
  Diese Funktion ist gedacht für *userLoginScripts*.
- `scriptWasExecutedBefore` //since 4.11.2.1
  Mit dieser boolschen Funktion kann überprüft werden, ob das *userLoginScript* zu diesem Produkt in dieser Version schon mal zuvor gelaufen ist und eine erneute Ausführung unnötig ist. Dazu liest diese Funktion zunächst einen evtl. vorhandenen Versionsstempel vom Profil ein (wie das mit `readVersionFromProfile` möglich ist) und vergleicht diesen mit der aktuell laufenden Version. Aus dem Vergleich ergibt sich der Rückgabewert (wahr/falsch). Danach werden noch die aktuellen Werte in das Profil zurückgeschrieben (wie das mit `saveVersionToProfile` möglich ist). Somit benötigen Sie nur diese eine Funktion in einer `if` Anweisung, um zu prüfen ob das Script schon mal gelaufen ist.

- `isLoginScript` //since 4.11.2.1
  Diese boolsche Funktion liefert *true* wenn das aktuelle Script als *userLoginScript* läuft. Siehe auch: `GetScriptMode`

7.9 Bedingungsanweisungen (if-Anweisungen)

Die Ausführung einer oder mehrere Anweisungen kann in den primären Sektionen von der Erfüllung bzw. Nichterfüllung einer Bedingung abhängig gemacht werden.

Beispiel:

```
;Welche Windows-Version?
DefVar $MSVersion$

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

7.9.1 Allgemeine Syntaxregeln

Folgendes Schema der if-Anweisung ist ersichtlich:
`if` <Bedingung>
;eine oder mehrere Anweisungszeilen
`else`
;eine oder mehrere Anweisungszeilen
`endif`

Der `else`-Teil der Anweisung darf fehlen.
if-Anweisungen können geschachtelt werden. Das bedeutet, dass in der Anweisung nach einem `if` Satz (sowohl in dem `if` als auch im `else` Teil) eine weitere if-Anweisung folgen darf.

<Bedingungen> sind boolesche Ausdrücke, das heißt logische Ausdrücke, die entweder den Wert *wahr* oder den Wert *falsch* tragen können.

7.9.2 Boolesche Ausdrücke

Ein Vergleichsausdruck, welcher ein boolscher Ausdruck ist, sieht folgendermaßen aus:
<STRINGAUSDRUCK> <Vergleichszeichen> <STRINGAUSDRUCK>
An der Stelle <Vergleichszeichen> kann eins der folgenden Zeichen stehen:
`< <= = >= >`

Bei String-Vergleichen im *opsi-winst* wird Groß-/Kleinschreibung nicht unterschieden.

Ungleich muss mit einem `NOT()` Ausdruck umgesetzt werden, was weiter unten gezeigt wird.

Es gibt einen Vergleichsausdruck um zwei Strings wie (integer) Zahlen zu vergleichen. Wenn einer der Werte nicht in eine Zahl übertragen werden kann, wird ein Fehler ausgegeben.
Diese Zahlenvergleichsausdrücke haben die gleich Form wie die String-Vergleichsausdrücke, allerdings wird dem dem Vergleichszeichen ein INT vorangestellt:
<STRINGAUSDRUCK> `INT`<Vergleichszeichen> <STRINGAUSDRUCK>
So können Ausdrücke wie

```
if $Name1$ INT<= $Name2$
```

oder

```
if $Number1$ INT>= $Number2$
```

gebildet werden.

Boolesche Operator sind `AND`, `OR` und `NOT()` (Groß-/Kleinschreibung nicht unterschieden).
b1, b2 und b3 sind boolesche Ausdrücke, die sich zu kombinierten Ausdrücken verbinden lassen.
b1 `AND` b2
b1 `OR` b2
`NOT(` b3 `)`
Diese booleschen Ausdrücke zeigen dabei eine Konjunktion (AND), ein Disjunktion (OR) und eine Negation (NOT).

Ein boolescher Ausdruck kann in runden Klammer eingeschlossen werden (diese produziert dann einen neuen booleschen Ausdruck mit dem selben Wert).

Die allgemeinen Regel für boolesche Operatorenprioritäten ("and" vor "or") sind im Moment nicht implementiert. Ein Ausdruck mit mehr als einem Operator wird von links nach rechts interpretiert. Wenn also eine boolescher Ausdruck einen AND und OR Operator enthalten soll, müssen runde Klammern eingesetzt werden. So muss zum Beispiel explizit geschrieben werden
b1 `OR` (b2 `AND` b3)
oder
(b1 `OR` b2) `AND` b3
Das zweite Beispiel beschreibt, was ausgeführt werden würde, wenn keine runden Klammern gesetzt wäre – wohingegen die übliche Operatorenprioritäten so laufen würde wie in der ersten Zeile angezeigt.

Boolesche Operatoren können als spezielle boolesche Wertefunktionen eingesetzt werden (die Negation-Operatoren demonstrieren das sehr deutlich).

Es sind noch weitere boolesche Funktionen implementiert. Jeder Aufruf einer solchen Funktion begründet sich in einen booleschen Ausdruck:

- `FileExists`(<datei name>)
  Die Funktion gibt wahr zurück, wenn die genannte Datei oder das Verzeichnis existiert, ansonsten kommt die Antwort falsch.
- `FileExists32`(<datei name>) siehe Kapitel 64 Bit-Unterstützung
- `FileExists64`(<datei name>) siehe Kapitel 64 Bit-Unterstützung
- `FileExistsSysNative`(<datei name>) siehe Kapitel 64 Bit-Unterstützung
- `LineExistsIn`(<Zeile>, <Dateiname>)
  Die Funktion gibt wahr zurück, wenn die Textdatei <Dateiname> eine Zeile beinhaltet, die im ersten Parameter beschrieben ist (jeder Parameter ist ein String-Ausdruck). Anderenfalls (oder falls die Datei garnicht existiert) wird falsch zurückgegeben.
- `LineBeginning_ExistsIn`(<string>, <Dateiname>)
  Die Funktion gibt wahr zurück, wenn die Zeile <string> in der Textdatei <Dateiname> vorhanden ist (jeder Parameter ist ein String-Ausdruck). Anderenfalls (oder falls die Datei garnicht existiert) wird falsch zurückgegeben.
- `XMLAddNamespace`(<XMLfilename>, <XMLelementname>, <XMLnamespace>)
  Mit dieser Funktion wird eine XML Namensraum im ersten XML-Element-Tag mit dem vergebenen Namen definiert (falls noch nicht vorhanden). Er wird ausgegeben, wenn ein Name eingefügt wurde. Die *opsi-winst* XML-Patch-Sektion benötigt diese Namensraumdefinition. Der File muss so formatiert werden, dass das Element-Tag keine Zeilenumbrüche beinhaltet. Für ein Anwendungsbeispiel siehe im Kochbuch Kapitel "Einfügen einer Namensraumdefinition in eine XML-Datei".

- `XMLRemoveNamespace`(<XMLfilename>, <XMLelementname>, <XMLnamespace>)
  Mit dieser Funktion wird die Definition des XML Namensraum wieder entfernt. Es wird wahr ausgegeben, wenn eine Entfernung erfolgt ist. (Beispiel im Kochbuch siehe Kapitel "Einfügen einer Namensraumdefinition in eine XML-Datei").

- `HasMinimumSpace`(<Laufwerksname>, <Kapazität>)
  gibt true zurück, wenn mehr Platz als die geforderte <Kapazität> auf dem Laufwerk <Laufwerksname> vorhanden ist. <Kapazität> ist syntaktisch ebenso wie <Laufwerksname> ein String-Ausdruck. Die <Kapazität> kann als Nummer ohne genauere Bezeichnung (dann interpretiert als Bytes) oder mit einer näheren Bezeichnung wie "kB", "MB" oder "GB" ausgegeben werden (case sensitive).
  Anwendungsbeispiel:

```
if not (HasMinimumSpace ("%SYSTEMDRIVE%", "500 MB"))
  LogError "Es ist nicht genug Platz auf dem Laufwerk %SYSTEMDRIVE%, erforderlich sind 500 MB"
  isFatalError
endif
```

- `opsiLicenseManagementEnabled`
  gibt true zurück wenn das opsi System über ein anwendbares (freigeschaltetes) Lizenzmanagement verfügt.

```
if opsiLicenseManagementEnabled
   set $mykey$ = DemandLicenseKey ("pool_office2007")
else
   set $mykey$ = GetProductProperty("productkey","")
endif
```

- `runningAsAdmin`
  Boolsche Funktion welche *true* liefert wenn das laufende Script mit Administrativen Rechten ausgeführt wird.
  Seit 4.11.1.1

- `isLoginScript`
  Boolsche Funktion welche *true* liefert wenn das laufende Script über die opsi Erweiterung *User Profile Management* als *userLoginScript* läuft.
  Seit 4.11.2.1

- `contains`(<str>, <substr>)
  Boolsche Funktion welche *true* liefert wenn <substr> in <str> enthalten ist. Die Funktion arbeitet case sensitive.
  Seit 4.11.3

- `isNumber`(<str>)
  Boolsche Funktion welche *true* liefert wenn <str> einen ganzahligen Wert (integer) representiert.
  Seit 4.11.3

7.10 Include Kommandos

Achtung
Die Verwendung von Include Kommandos führt schnell zu unübersichtlichen Code.
Lassen Sie die Finger davon wenn Sie Anfänger sind.

7.10.1 Include Kommandos: Syntax

Mit Include Kommandos kann der Inhalt einer externen Datei dem laufende Script hinzugefügt werden. Dies kann entweder einfügend oder anhängend erfolgen. Die Include Kommandos sind normale Kommandos der primären Sektionen. Die eingefügten Dateien können weitere Include Kommandos enthalten.
Diese Kommandos gibt es seit Version 4.11.3

- `include_insert` <file name>
  Fügt den Inhalt von <file name> nach der aktuellen Zeile im laufenden Script ein. Somit ist die erste Zeile der eingefügten Datei die nächste Zeile welche der *opsi-winst* interpretiert.
- `include_append` <file name>
  Fügt den Inhalt von <file name> am Ende des laufenden Scriptes ein. Diese Anweisung dient vor allem dazu Sektionen aus z.B. einer Bibliothek hinzu zufügen.

Für beide Funktionen gilt:
<file name> ist:

- Ein kompletter Pfad zu einer Datei.
- Eine Datei in `%ScriptPath%`
- Eine Datei in `%opsiScriptHelperPath%\lib`
  Entspricht: *%ProgramFiles32Dir%\opsi.org\opsiScriptHelper\lib*
- Eine Datei in `%WinstDir%\lib`

Die Prüfung erfolgt in dieser Reihenfolge. Die erste Datei die gefunden wird, wird genommen.

Beispiel:
Wir haben folgendes Script:

```
[Actions]
include_append "section_Files_del_tmp_dummy.opsiinc"
include_insert "include-test1.opsiinc"
```

Dabei hat die Datei `include-test1.opsiinc` folgenden Inhalt:

```
DefVar $inctestvar$
set $inctestvar$ = "inctest"
Files_del_tmp_dummy
include_append "section_Files_copy_inctest.opsiinc"
Files_copy_inctest

if fileExists("c:\opsi.org\tmp\dummy.txt")
        comment "passed"
else
        comment "failed"
        set $TestResult$ = "not o.k."
        LogWarning "include test failed"
endif

if fileExists("%scriptpath%\test-files\dummy.txt")
        comment "passed"
else
        comment "failed"
        set $TestResult$ = "not o.k."
        LogWarning "include test failed"
endif
Files_del_tmp_dummy
```

Dabei hat die Datei `section_Files_copy_inctest.opsiinc` folgenden Inhalt:

```
[Files_copy_inctest]
copy "%scriptpath%\test-files\dummy.txt" "c:\opsi.org\tmp"
```

Dabei hat die Datei `section_Files_del_tmp_dummy.opsiinc` folgenden Inhalt:

```
[Files_del_tmp_dummy]
del -f "c:\opsi.org\tmp\dummyt.txt"
```

7.10.2 Include Kommandos: Library

Mit der Version 4.11.3 werden folgende Includefiles in `%WinstDir%\lib` ausgeliefert:

`insert_check_exit_code.opsiinc`:

```
; opsi include file

DefVar $ExitCode$

include_append "section_sub_check_exitcode.opsiinc"
```

`insert_get_licensekey.opsiinc`:

```
; opsi include file

DefVar $LicenseRequired$
DefVar $LicenseKey$
DefVar $LicensePool$

include_append "section_sub_get_licensekey.opsiinc"
```

`section_sub_check_exit_code.opsiinc`:

```
;opsi include file

[Sub_check_exitcode]
comment "Test for installation success via exit code"
set $ExitCode$ = getLastExitCode
; informations to exit codes see
; http://msdn.microsoft.com/en-us/library/aa372835(VS.85).aspx
; http://msdn.microsoft.com/en-us/library/aa368542.aspx
if ($ExitCode$ = "0")
        comment "Looks good: setup program gives exitcode zero"
else
        comment "Setup program gives a exitcode unequal zero: " + $ExitCode$
        if ($ExitCode$ = "1605")
                comment "ERROR_UNKNOWN_PRODUCT  1605    This action is only valid for products
    that are currently installed."
                comment "Uninstall of a not installed product failed - no problem"
        else
                if ($ExitCode$ = "1641")
                        comment "looks good: setup program gives exitcode 1641"
                        comment "ERROR_SUCCESS_REBOOT_INITIATED 1641    The installer has
    initiated a restart. This message is indicative of a success."
                        ExitWindows /Reboot
                else
```

```
                        if ($ExitCode$ = "3010")
                                comment "looks good: setup program gives exitcode 3010"
                                comment "ERROR_SUCCESS_REBOOT_REQUIRED  3010    A restart is
    required to complete the install. This message is indicative of a success."
                                ExitWindows /Reboot
                        else
                                logError "Fatal: Setup program gives an unknown exitcode unequal
    zero: " + $ExitCode$
                                isFatalError "Exit Code: "+ $ExitCode$
                        endif
                endif
        endif
endif
```

section_sub_get_licensekey.opsiinc:

```
; opsi include file

[Sub_get_licensekey]
if opsiLicenseManagementEnabled
        comment "License management is enabled and will be used"

        comment "Trying to get a license key"
        Set $LicenseKey$ = demandLicenseKey ($LicensePool$)
        ; If there is an assignment of exactly one licensepool to the product the following call
    is possible:
        ; Set $LicenseKey$ = demandLicenseKey ("", $ProductId$)
        ;
        ; If there is an assignment of a license pool to a windows software id, it is possible to
     use:
        ; DefVar $WindowsSoftwareId$
        ; $WindowsSoftwareId$ = "..."
        ; Set $LicenseKey$ = demandLicenseKey ("", "", $WindowsSoftwareId$)

        DefVar $ServiceErrorClass$
        set $ServiceErrorClass$ = getLastServiceErrorClass
        comment "Error class: " + $ServiceErrorClass$

        if $ServiceErrorClass$ = "None"
                comment "Everything fine, we got the license key '" + $LicenseKey$ + "'"
        else
                if $ServiceErrorClass$ = "LicenseConfigurationError"
                        LogError "Fatal: license configuration must be corrected"
                        LogError getLastServiceErrorMessage
                        isFatalError $ServiceErrorClass$
                else
                        if $ServiceErrorClass$ = "LicenseMissingError"
                                LogError "Fatal: required license is not supplied"
                                isFatalError $ServiceErrorClass$
                        endif
                endif
        endif
else
        LogError "Fatal: license required, but license management not enabled"
        isFatalError "No Licensemanagement"
endif
```

7.11 Aufrufe von Unterprogrammen

Anweisungen in primären Sektionen, die auf einen Programmtext an anderer Stelle verweisen, sollen hier Unterprogramm- oder Prozeduraufrufe heißen.

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

So "ruft" in obigem Beispiel die Anweisung in der `Actions`-Sektion

```
sub_install_winXP
```

die Sektion *[sub_install_winXP]*, welche dann im Skript an anderer Stelle nachzulesen ist als

```
[sub_install_winXP]
Files_Kopieren_XP
WinBatch_SetupXP
```

Weil es sich in diesem Beispiel um eine `Sub`-Sektion handelt, also immer noch um eine primäre Sektion, kann in ihr wiederum auf weitere Sektionen verwiesen werden, in diesem Fall auf die Sektionen *[Files_Kopieren_XP]* und *[WinBatch_Setup_XP]*.

Generell gibt es drei Wege um die genannten Anweisungen zu platzieren:

1. Der gebräuchlichste Ort für den Aufruf eines `Sub`-Sektion ist eine weitere interne Sektion im Skript, wo die aufgerufene Befehle platziert werden (wie in dem Beispiel).
2. Die bezeichneten Befehle können auch in einer andere Datei untergebracht werden, welche als externe Sektion läuft.
3. Jede String-Liste kann als eine Befehlsliste für einen Sub-Programm Aufruf benutzt werden.

Zur Syntax der Sub-Programm Aufrufe im einzelnen:

7.11.1 Komponenten eines Unterprogrammaufrufs

Formal kann die Syntax wie folgt aufgerufen werden
<proc. type>(<proc. name> | <External proc. file> | <Stringlisten Funktion>)

Diese Ausdrücke können durch einen oder mehrere Parameter ergänzt werden (ist vom Ablauftyp abhängig).

Das bedeutet: Ein Ablauf besteht aus drei Hauptbereichen.

Der erste Teil ist der Unterprogramm Typnamen.
Beispiele für Typennamen sind `Sub` (Aufruf einer primären Sektion bzw. eines Unterprogramms des primären Typs) sowie `Files` und `WinBatch` (diese Aufrufe sind speziell für die zweite Sektion).
Den kompletten Überblick über die existierenden `Sub`-Programmtypen sind am Anfang von Kapitel "Aufrufe von Unterprogrammen" genauer beschrieben.

Der zweite Teil bestimmt, wo und wie die Zeilen des Subprogramms gefunden werden. Dazu gibt es zwei Möglichkeiten:

1. Das Sub-Programm ist eine Zeilenabfolge, die im sich ausführbaren Bereich des *opsi-winst* Skripts als interne Sektion befindet. Es wird ein eindeutiger Sektionsname (bestehend aus Buchstaben, Zahlen und einigen Sonderzeichen) hinzugefügt, um den Programmtyp näher zu beschreiben (ohne Leerzeichen).
   z.B.
   sub_install_winXP
   oder
   files_copy_winXP
   Sektionsnamen sind case insensitive wie jeder andere String.

2. Wenn der Programmtyp alleine steht, wird eine String-Liste oder ein String-Ausdruck erwartet. Wenn der folgende Ausdruck nicht als String-Listenausdruck aufgelöst werden kann (vgl. 3.) wird ein String-Ausdruck erwartet. Der String wird dann als Dateiname interpretiert. Der *opsi-winst* versucht die Datei als Textdatei zu öffnen und interpretiert die Zeilen als eine externe Sektion des beschriebenen Typs.
   Bsp.:
   `sub` *"p:\install\opsiutils\mainroutine.ins"*
   Es wird versucht die Zeile mainroutine.ins als Anweisung der Subsektion auszulesen.

3. Wenn der Ausdruck auf eine alleinstehenden spezifizierten Sektionstyp folgt, kann dieser als ein String-Listenausdruck aufgelöst werden. Die String-Listenkomponenten werden dann als ein Sektionsausdruck interpretiert.
   Dieser Mechanismus kann bspw. dazu verwendet werden, um eine Datei mit Unicode-Format zu laden und dann mit den üblichen Mechanismen zu bearbeiten:

```
registry loadUnicodeTextFile("%scriptpath%/opsiorgkey.reg") /regedit
```

Syntaktisch hat diese Zeile die drei Bestandteile:
* `registry`, die eigentliche Anweisung, die den Sektionstyp spezifiziert.
* `loadUnicodeTextFile (..)`, ein String-Listenausdruck, in dem näher beschrieben ist, wie man eine Zeile der `registry` Sektion bekommt.
* `/regedit`, Option als 2. Parameter (typspezifisch, s. das Folgende).

In diesem Beispiel gibt der Aufrufparameter ein Beispiel an für den dritten Teil eines Subsektionsaufrufs:

Der dritte Part eine Aufrufs umfasst spezielle Aufrufsoptionen. Referenzen für die Aufrufsoptionen beziehungsweise für eine genauere Beschreibung der Sektionsaufrufe finden sich in siehe Kapitel "Sekundäre Sektionen".

7.12 Reboot-Steueranweisungen

Die Anweisung `ExitWindows` dient zur Steuerung von Reboot, Shutdowns, u.ä. Vorgängen welche erst nach Beendigung des *opsi-winst* selbst durchgeführt werden. Die Benennung des Befehls und die Tatsache, das es `ExitWindows` nicht ohne Modifier gibt, ist historisch bedingt: Unter Windows 3.1 konnte man Windows beenden und zur DOS-Ebene zurück wechseln.

- `ExitWindows /RebootWanted`
  Abgekündigt: vermerkt eine Rebootanfrage eines Skriptes in der Registry, lässt aber das *opsi-winst* Skript weiterlaufen und weitere Skripte abarbeiten und rebootet erst, wenn alle Skripte durchgelaufen sind. Eigentlich wird dieses Kommando jetzt als `ExitWindows /Reboot` behandelt (da ansonsten eine Installation fehlschlagen könnte, weil ein benötigtes Produkt nicht komplett installiert wurde).

- `ExitWindows /Reboot`
  unterbricht eine Skriptfolge durch die Auslösung des Reboots nachdem der *opsi-winst* die Bearbeitung des laufenden Skripts beendet hat.

- `ExitWindows /ImmediateReboot`
  unterbricht die normale Ausführung eines Skripts, an der Stelle, an der er aufgerufen wird. Nach dem der Befehl aufgerufen wurde, werden (außer if-Anweisungen) keine Anweisungen mehr ausgeführt und der Rechner rebootet.

Dabei bleibt in der opsi-Umgebung der Actionrequest der das Skript aufgerufen hat bestehen.Dadurch wird gewährleistet, dass nach dem Neustart der *opsi-winst* wieder das Skript, dass abgebrochen wurde, startet. Das Skript sollte daher so konzipiert sein, dass die Ausführung nach dem Punkt der Unterbrechung fortgesetzt wird (andernfalls erhalten wir möglicherweise eine Endlosschleife. . .) vgl. das Beispiel in diesem Abschnitt.

- `ExitWindows /ImmediateLogout`
  funktioniert ähnlich wie der Aufruf /ImmediateReboot, aber beinhaltet ein Beenden des *opsi-winst* (Beenden des Skriptes) statt einen Reboot. Dies ist dann sinnvoll, wenn nachfolgend ein automatisches Login (eines anderen Users) folgen soll. Beachten Sie hierzu Kapitel "Skript für Installationen im Kontext eines lokalen Administrators".
- `ExitWindows /ShutdownWanted`
  sorgt dafür, dass der PC nach Abschluss der Installation aller angefragten Produkte heruntergefahren wird.

Wie man eine Markierung setzt, um sicherzustellen, dass das Skript nicht in eine Endlosschleife läuft, wenn `ExitWindows /ImmediateReboot` aufgerufen wird, demonstriert folgendes Codebeispiel:

```
DefVar $Flag$
DefVar $WinstRegKey$
DefVar $RebootRegVar$

Set $WinstRegKey$ = "HKLM\SOFTWARE\opsi.org\winst"
Set $Flag$ = GetRegistryStringValue32("["+$WinstRegKey$+"] "+"RebootFlag")

if not ($Flag$ = "1")
  ;=========================
  ; Anweisungen vor Reboot

  Files_doSomething

  ; Reboot initialisieren ...
  Set $Flag$ = "1"
  Registry_SaveRebootFlag
  ExitWindows /ImmediateReboot

else
  ;=========================
  ; Anweisungen nach Reboot

  ; Rebootflag zurücksetzen
  Set $Flag$ = "0"
  Registry_SaveRebootFlag

  ; die eigentlichen Anweisungen

  Files_doMore

endif


[Registry_SaveRebootFlag]
openKey [$WinstRegKey$]
set "RebootFlag" = "$Flag$"

[Files_doSomething]
; eine Sektion, die vor dem Reboot ausgeführt wird

[Files_doMore]
; eine Sektion, die nach dem Reboot ausgeführt wird
```

7.13 Fehlgeschlagene Installation anzeigen

Passieren bei einer Installation Fehler, die zum Fehlschlagen der Installation führen, so sollte dies an den Server zurückgemeldet werden.

Um in einem *opsi-winst* Skript, eine Installation als gescheitert zu erklären, gibt es eine Ausdruck namens
`isFatalError`
unterbricht die normale Ausführung eines Skripts, an der Stelle, an der er aufgerufen wird. Nach dem der Befehl aufgerufen wurde, werden (außer if-Anweisungen) keine Anweisungen mehr ausgeführt und als Skriptergebnis wird *failed* zurückgeliefert. Wird dieser Befehl nicht aufgerufen, so ist das Skriptergebnis *success*.

Seit 4.11.3.2 ist auch die folgende Variante erlaubt:

- `isFatalError` <string>
  wobei <string> als kurze Fehlerbeschreibung an den opsi-server als *actionProgress* weitergegeben wird und im opsi-configed angezeigt wird.

Es gibt **keinen** Automatismus innerhalb eines Winst-Skriptes, um zu einen *failed* Ergebnis zu kommen. Sie müssen skriptgesteuert den Fehler selbst feststellen. Hierzu gibt Ihnen der *opsi-winst* einige Hilfsmittel.

Ein „fataler Fehler“ sollte zum Beispiel ausgelöst werden, wenn der Plattenplatz für die Installation nicht ausreicht:

```
DefVar $SpaceNeeded$"
Set $SpaceNeeded$" = "200 MB"

if not(HasMinimumSpace ("%SYSTEMDRIVE%", $SpaceNeeded$"))
  LogError "Nicht genügend Platz. Erforderlich sind "+$SpaceNeeded$
  isFatalError
  ; beendet die Skriptausführung und setzt den Produktstaus auf failed
else
  ; die Installation wird gestartet
  ; ...
endif
```

Fehler die von Funktionen des *opsi-winst* zurückgeliefert werden, werden in die Logdatei geschrieben und erhöhen den Fehlerzähler des opsi-winst. Dieser Fehlerzähler kann ausgewertet werden. So besteht auch die Möglichkeit, in einem kritischen Abschnitt eines Skripts festzustellen, ob Fehler bzw. wie viele Fehler aufgetreten sind (und abhängig hiervon ggf. isFatalError aufzurufen).

Dafür ist die Fehlerzählung zu Beginn des entsprechenden Abschnittes – z.B. vor einer Files-Sektion – mit
`markErrorNumber`
zu initialisieren. Die Zahl der Fehler, die ab dieser Stelle aufgetreten sind, kann dann mit dem Ausdruck
`errorsOccuredSinceMark`
abgefragt werden. Z.B. kann man die Bedingung „es kam in diesem Abschnitt mindestens ein Fehler vor“ so formulieren:
`if errorsOccuredSinceMark > 0`
und, wenn es sinnvoll erscheint, daraufhin
`isFatalError`
feststellen.

Sofern die Skriptanweisungen nicht direkt einen Fehler produzieren, jedoch aufgrund bestimmter Umstände eine Situation trotzdem als Fehlersituation gewertet werden soll, kann auch mittels der Anweisung `logError` eine Fehlermeldung generiert werden.

```
markErrorNumber
; Fehler, die nach dieser Marke auftreten werden gezählt
; und werden als fatale Fehler gewertet

logError "test error"
; wir schreiben einen Kommentar "test error" in die Logdatei
```

```
; und die Fehleranzahl wird um eins erhöht
; für Testzwecke kann man diese Zeile auskommentieren

if errorsOccuredSinceMark > 0
    ; die Skriptausführung wird so bald wie möglich beendet
    ; und setzt den Produktstatus auf "failed"

    isFatalError
    ; Kommentare können  noch geschrieben werden

    comment "error occured"

else
    ; kein Fehler aufgetreten, gibt folgendes aus:

    comment "no error occured"
endif
```

Kapitel 8

Sekundäre Sektionen

Sekundäre Sektionen können ebenso wie die primäre Sektion aufgerufen werden, haben aber eine andere Syntax. Die Syntax der verschiedenen Sektionstypen ist jeweils aufgabenbezogen und lehnt sich möglichst eng an bekannte Kommandoformen für den jeweiligen Aufgabentyp an.

Sekundäre Sektionen sind spezifiziert für einen eingegrenzten Funktionsbereich. Dies bezieht sich auf das Objekt der Funktion z.B. das Dateisystem im Allgemeinen, die Windows Registry oder XML-Dateien. Einen spezielle Bedeutung haben die verschiedenen Varianten der Batch-Sektionen, die dazu dienen (beliebige) externe Programme oder Skripte aufzurufen.

8.1 Files-Sektionen

Files-Sektion dienen zum Dateihandling (Kopieren, Löschen). Anders als bei Ausführung der vergleichbaren Kommandozeilen-Befehle, werden die ausgeführten Operationen bei Bedarf genau protokolliert. Zusätzlich kann beim Kopieren von Bibliotheksdateien (z.B. dll-Dateien) auch die Dateiversion geprüft werden, so dass nicht neuere Versionen durch ältere Versionen überschrieben werden.

8.1.1 Beispiele

Eine Files-Sektion könnte etwa lauten:

```
[Files_do_some_copying]
copy -sV "p:\install\instnsc\netscape\*.*" "C:\netscape"
copy -sV "p:\install\instnsc\windows\*.*" "%SYSTEMROOT%"
```

Mit diesen Anweisungen werden alle Dateien des Verzeichnisses *p:\install\instnsc\netscape* in das Verzeichnis *C:\netscape* kopiert, sowie alle Dateien von *p:\install\instnsc\windows* in das Windows-Systemverzeichnis (welches das jeweils Gültige ist, steht automatisch in der opsi-winst-Konstante %SYSTEMROOT%). Die Option `-s` bedeutet dabei, das alle Subdirectories mit erfasst werden, `-V` steht für das Einschalten der Versionskontrolle von Bibliotheksdateien.

8.1.2 Aufrufparameter (Modifier)

In der Regel benötigt eine Files-Sektion beim Aufruf keinen Parameter.

Es gibt jedoch eine spezielle Anwendung der Files-Sektion, bei der Zielpfad von Kopieraktionen automatisch bestimmt oder modifiziert wird, bei denen die betreffende Files-Sektion mit dem Parameter

- `/AllNTUserProfiles` bzw.

- `/AllNTUserSendTo`

aufgerufen wird.
Beide Varianten bedeuten:
Die betreffende Files-Sektion wird je einmal für jeden NT-User ausgeführt. Bei Kopieraktionen in dieser Files-Sektion wird automatisch ein User-spezifisches Verzeichnis als Zielverzeichnis eingesetzt.

Für alle anderen Aktionen steht eine Variable `%UserProfileDir%` oder seit *opsi-winst* version 4.11.2 `%CurrentProfileDir%` zur Verfügung, mit der Verzeichnisnamen konstruiert werden können.

Das User-spezifische Verzeichnis ist im Fall von `/AllNTUserProfiles` das User-Profilverzeichnis. Im Fall von `/AllNTUserSendTo` wird der Pfad zum User-spezifischen SendTo-Ordner vorgegeben (der dazu dient, im Windows-Explorer Kontextmenü-Verknüpfungen vorzugeben).

Die genaue Regel, nach der Zielpfade für copy-Anweisungen automatisch gebildet werden, ist dreiteilig:

1. Folgt auf die Angabe der zu kopierenden Quelldateien gar keine Zielverzeichnisangabe, so werden die Dateien direkt in das betreffende User-spezifische Verzeichnis kopiert. Der Befehl lautet einfach
   `copy` <Quelldatei>
   Dies ist gleichbedeutend mit
   `copy` <Quelldatei> *"%UserProfileDir%"*
   oder seit 4.11.2
   `copy <source file(s)> "%CurrentProfileDir%"`
2. Wenn außer den zu kopierenden Quelldateien ein Kopierziel *targetdir* spezifiziert ist, dieses Ziel jedoch keinen absoluten Pfad (beginnend mit Laufwerksname oder mit "\") darstellt, dann wird die Zielangabe als Unterverzeichnisname des User-spezifischen Verzeichnisses interpretiert und der Zielverzeichnisname dementsprechend konstruiert. D.h. man schreibt
   `copy` <Quelldateien> *targetdir* und das wird interpretiert wie:
   `copy` <Quelldatei> *"%UserProfileDir%\targetdir"*
   oder seit 4.11.2
   `copy <source file(s)> "%CurrentProfileDir%\targetdir"`

Die Verwendung von `%CurrentProfileDir%` hat den Vorteil, das die selbe *Files* Sektion mit `/AllNTUserProfiles` verwendet werden kann wenn das Script nicht als *userLoginScript* (in *Machine* script mode) läuft und ohne `/AllNTUserProfiles` wenn das Script als *userLoginScript* läuft (in *Login* script mode).

1. Enthält der copy-Befehl Quelldateien und einen absoluten Zielpfad targetdir, so wird dieser statische Zielpfad verwendet.

Weiterhin gibt es die Aufrufparameter:

- `/32Bit`
- `/64Bit`
- `/SysNative`

Welche die *file redirection* auf 64 Bit-Systemen beeinflussen. siehe Kapitel 64 Bit-Unterstützung

8.1.3 Kommandos

Innerhalb einer Files-Sektion sind die Anweisungen

- `Copy`
- `Delete`
- `SourcePath`

- `CheckTargetPath`

definiert.

Die Kommandos `Copy` und `Delete` entsprechen im Wesentlichen den Windows-Kommandozeilen-Befehlen xcopy bzw. del.

`SourcePath` sowie `CheckTargetPath` legen Quell- bzw. Zielpfad einer Kopieraktion fest, ähnlich wie sie etwa im Windows-Explorer durch Öffnen von Quell- und Zieldirectory in je einem Fenster realisiert werden kann. Der Zielpfad wird, sofern er nicht existiert, erzeugt.

Im Einzelnen:

- `Copy` [-svdunxwnr] <Quelldatei(maske)> <Zielpfad>

  Die Quelldateien können dabei mittels Joker ("* " in der Dateimaske) oder auch nur mit einer Pfadangabe bezeichnet werden.

  

  Achtung
  Zielpfad wird in jedem Fall als Directory-Name interpretiert. Umbenennen beim Kopieren ist nicht möglich: Ziel ist immer ein Pfad, nicht ein (neuer) Dateinamen. Existiert der Pfad nicht, wird er (auch mit geschachtelten Directories) erzeugt.

  Die einzelnen (in beliebiger Reihenfolge aufführbaren) Optionen der Copy-Anweisung bedeuten:

  - `s` → Mit Rekursion in Subdirectories.
  - `e` → Leere (Empty) Subdirectories.
    Gibt es leere Subdirectories im Quellverzeichnis, werden sie im Zielverzeichnis ebenfalls leer ("empty") erzeugt.
  - `V` → Mit Versionskontrolle
    Mit Versionskontrolle:
    Neuere Versionen von Windows-Bibliotheksdateien im Zielverzeichnis werden nicht durch ältere Versionen überschrieben (bei unklaren Verhältnissen wird in einem Log-Eintrag gewarnt).
  - `v` → (nicht Verwenden)
    Mit Versionskontrolle:
    Veraltet; bitte nicht bei Betriebssystemversionen höher als Win2k verwenden, da hier nicht nur gegen das Zielverzeichnis, sondern auch gegen %SYSTEM% geprüft wird. Verwenden Sie stattdessen `-V`.
  - `d` → Mit Datumskontrolle:
    Jüngere *.EXE-Dateien werden nicht durch ältere überschrieben.
  - `u` → Datei-Update:
    Es werden Dateien nur kopiert, sofern sie nicht mit gleichem oder jüngerem Datum im Zielpfad existieren.
  - `x` → x-tract
    Wenn eine Datei ein Zip-Archiv ist, wird es entpackt (x-tract). Vorsicht: Zip-Archive verbergen sich unter verschiedenen Dateinamen (z.B. sind Java jar-Dateien auch Zip-Archive), daher sollte man die Extract-Option nicht unbesehen auf alle Dateien anwenden. Achtung: Es werden keine Pfade entpackt.
  - `w` → weak
    Dateien werden nur überschrieben, wenn sie keinen Schreibschutz haben (das Überschreiben ist "weak" (relativ schwach) im Vergleich zum Defaultverhalten, dem Ignorieren des Schreibschutzes).
  - `n` → no over write
    Dateien werden nicht überschrieben.
  - `c` → continue
    Wenn eine Systemdatei in Benutzung ist, kann sie erst nur nach einem Reboot überschrieben werden. Das *opsi-winst* default-Verhalten ist dabei, dass ein Datei in Benutzung zum Überschreiben beim nächsten Reboot markiert wird UND die *opsi-winst* Reboot Markierung gesetzt wird. Das Setzen der Kopiermodifikation "-c" stellt den automatischen Reboot aus. Anstatt das normale Prozesse weiterlaufen (continues) wird das Kopieren nur vervollständigt, wenn ein Reboot auf eine andere Weise ausgelöst wird.

  - `r` → read-only Attribute
    Nur wenn diese Option gesetzt ist, bleibt ein eventuell vorhandenes read-only-Attribut erhalten (im Gegensatz zu dem default-Verhalten, welches read-only Attribute ausschaltet).

- `Delete` [-sfd[n]] <Pfad>

oder

- `Delete` [-sfd[n]] <Datei(maske)>

  Löschen einer Datei bzw. eines Verzeichnisses. Mögliche Optionen (die in beliebiger Reihenfolge aufgeführt sein können) sind:

  - `s` → subdirectories
    Steht für die Rekursion in Subdirectories, das heißt, der ganze Pfad bzw. alle der Dateimaske entsprechenden Dateien im Verzeichnisbaum ab der angegebenen Stelle werden gelöscht.

    

    Achtung
    Der Befehl
    `delete -s c:\opsi`
    Bedeutet nicht lösche das Verzeichnis *c:\opsi* rekursiv, sondern lösche ab *c:* rekursiv alle Dateien namens *opsi* (und führt damit evtl. zum kompletten Durchsuchen der Festplatte). Zum rekursiven Löschen von *c:\opsi* verwenden Sie das Kommando:
    `delete -s c:\opsi\`
    Durch den angehängen Backslash ist deutlich, dass Sie ein Verzeichnis meinen.
    Es ist sicherer das Kommando `del` stattdessen zu verwenden

  - `f` → force
    Erzwingt ("force") das Löschen auch von read-only-Dateien.
  - `d [n]` → date
    Dateien werden nur gelöscht, sofern sie mindestens n Tage alt sind. Default für n ist 1.

- `del` [Options] <path[/mask]] //since 4.11.2.1
  Arbeitet wie `delete` aber bei
  `del -s -f c:\not-exists`
  wenn `c:\not-exists` nicht existiert wird nicht das komplette `c:\` nach `not-exits` durchsucht.

Beispiel (**Der anhängende Backslash darf weggelassen werden**):
`del -sf c:\delete_this_dir`

- `SourcePath` = <Quelldirectory>
  Festlegung des Verzeichnisses <Quelldirectory> als Vorgabe-Quelldirectory für in der betreffenden Sektion folgende Copy- sowie (!) Delete-Aktionen.

- `CheckTargetPath` = <Zieldirectory>
  Festlegung des Verzeichnisses <Zieldirectory> als Vorgabe-Zieldirectory für Copy-Aktionen. Wenn <Zieldirectory> nicht existiert, wird der Pfad auch mehrstufig erzeugt.

8.2 Patches-Sektionen

Eine Patches-Sektion dient der Modifikation (dem "Patchen") einer "*.INI-Datei", d.h. einer Datei, die Sektionen mit Einträgen der Form *<Variable> = <Wert>* besteht. Die Sektionen oder Abschnitte sind dabei gekennzeichnet durch Überschriften der Form *[Sektionsname]*.

(Seitdem eine gepatchete INI-Datei auf die gleiche Weise wie die Sektionen vom *opsi-winst* Skript erstellt werden, muss man vorsichtig mit den Bezeichnungen umgehen, damit kein Durcheinander entsteht).

8.2.1 Beispiele

```
Patches_DUMMY.INI $HomeTestFiles$+"\dummy.ini"

[Patches_dummy.ini]
add [secdummy] dummy1=add1
; werden durch andere Funktionen ueberschrieben
add [secdummy] dummy2=add2
add [secdummy] dummy3=add3
add [secdummy] dummy4=add4
add [secdummy] dummy5=add5
add [secdummy] dummy6=add6
set [secdummy] dummy2=set1
addnew [secdummy] dummy1=addnew1
change [secdummy] dummy3=change1
del [secdummy] dummy4
Replace dummy6=add6 replace1=replace1
```

ergibt folgenden Log:

```
Execution of Patches_DUMMY.INI
      FILE C:\tmp\testFiles\dummy.ini
      Info: This file does not exist and will be created
  addEntry [secdummy] dummy1=add1
    addSection [secdummy]
      done
      done
  addEntry [secdummy] dummy2=add2
      done
  addEntry [secdummy] dummy3=add3
      done
  addEntry [secdummy] dummy4=add4
      done
  addEntry [secdummy] dummy5=add5
      done
  addEntry [secdummy] dummy6=add6
      done
  setEntry [secdummy] dummy2=set1
    Entry      dummy2=add2
    changed to dummy2=set1
  addNewEntry [secdummy] dummy1=addnew1
    appended entry
  changeEntry [secdummy] dummy3=change1
    entry      dummy3=add3
    changed to dummy3=change1
  delEntry [secdummy] dummy4
    in section secdummy deleted  dummy4=add4
  replaceEntrydummy6=add6 replace1=replace1
    replaced in line 7
  C:\tmp\testFiles\dummy.ini saved back
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_patches$ = "on"*

8.2.2 Aufrufparameter

Der Name der zu patchenden Datei wird als Parameter übergeben.

Als optionalen Modifier gibt es:

- `/AllNTUserProfiles`
  Wird eine Patches Sektion mit diesem Modifier aufgerufen und der Pfad zur zu patchenden Datei enthält die Konstante `%UserProfileDir%`, so wird diese Patchsektion für alle Profile ausgeführt.
  Eine *Patches* Sektion welche in einer `[ProfileActions]` Sektion aufgerufen wird hat im *Machine* Modus den Modifier `/AllNTUserProfiles` implizit. Im Loginscript Modus wird dann `%UserProfileDir%` als `%CurrentProfileDir%` interpretiert.
  (Seit Version 4.11.3.2)

8.2.3 Kommandos

In einer Patches-Sektion sind die Anweisungen

- `add`
- `set`
- `addnew`
- `change`
- `del`
- `delsec`
- `replace`

definiert. Eine Anweisung bezieht sich jeweils auf eine Sektion der zu patchenden Datei. Der Name dieser Sektion steht in Klammern `[]`.

Syntax und Funktion der Anweisungen im Einzelnen:

- `add` [<section name>] <variable1> `=` <value1>
  Fügt einen Eintrag der Form <Variable1> = <value1> in die Sektion <section name> ein, falls dort noch kein Eintrag von <Variable1> (auch mit anderem Wert) existiert. Im anderen Fall wird nichts geschrieben. Existiert die Sektion noch nicht, wird sie zuerst erzeugt.
- `set` [<section name>]<variable1> `=` <value1>
  Setzt einen vorhandenen Eintrag <variable1> = <value X> in der Sektion <section name>, um auf <variable1> = <value1> zu kommen. Existieren mehrere Einträge von <variable1>, wird der Erste umgesetzt. Falls kein Eintrag mit <variable1> existiert, wird <variable1> = <value1> in der Sektion <section name> erzeugt; existiert die Sektion noch nicht, wird sie zuerst erzeugt.
- `addnew` [<section name>]<variable1> `=` <value1>
  Der Eintrag <variable1> = <value1> wird in der Sektion <section name> auf jeden Fall erzeugt, sofern er dort nicht bereits genau so existiert (gegebenenfalls zusätzlich zu anderen Einträgen von <variable1>). Existiert die Sektion noch nicht, wird sie zuerst erzeugt.
- `change` [<section name>]<variable1> `=` <value1>
  Ändert einen vorhandenen Eintrag von <variable1> in der Sektion <section name> auf <variable1> = <value1>. Falls <variable1> nicht vorhanden ist, wird nichts geschrieben.
- `del` [<section name>] <variable1> `=` <value1>
  bzw.
  `del` [<section name>] <variable1>
  In der Sektion <section name> wird gemäß dem ersten Syntaxschema der Eintrag <variable1> = <value1> entfernt. Nach dem zweiten Syntaxschema wird der erste Eintrag von <variable1> aus der angesprochenen Sektion gelöscht, unabhängig vom Wert des Eintrags.

- `delsec [<section name>]`
  Die Sektion <section name> der .INI-Datei wird mitsamt ihren Einträgen gelöscht.
- `replace` <variable1>=<value1> <variable2>=<value2>
  In allen Sektionen der .INI-Datei wird der Eintrag <variable1>=<value1> durch <Variable2>=<value2> ersetzt. Zur Anwendung dieses Befehls dürfen Leerzeichen weder um die Gleichheitszeichen stehen noch in <value1> bzw. <value2> enthalten sein.

8.3 PatchHosts-Sektionen

Eine PatchHosts-Sektion dient der Modifikation einer *hosts*-Datei, das heißt einer Datei, deren Zeilen nach dem Schema *ipAdresse Hostname Aliasname(n) # Kommentar*
aufgebaut sind.

Dabei sind *Aliasname(n)* und *Kommentar* optional. Eine Zeile kann auch mit dem Symbol # beginnen und ist dann insgesamt ein Kommentar.

Die zu patchende Datei kann als Parameter des `PatchHosts`-Aufrufs angegeben sein. Fehlt der Parameter *HOSTS*, so wird in den Verzeichnissen (in dieser Reihenfolge) *c:\nfs, c:\windows* sowie *%systemroot%\system32\drivers\etc* nach einer Datei mit dem Namen *hosts* gesucht.

Wird auf keine dieser Arten eine Datei mit dem Namen *hosts* gefunden, bricht die Bearbeitung mit einem Fehler ab.

In einer PatchHosts-Sektion existieren die Anweisungen

- `setAddr`
- `setName`
- `setAlias`
- `delAlias`
- `delHost`
- `setComment`

Beispiel:

```
PatchHosts_add $HomeTestFiles$+"\hosts"

[PatchHosts_add]
setAddr ServerNo1 111.111.111.111
setName 222.222.222.222 ServerNo2
setAlias ServerNo1 myServerNo1
setAlias 222.222.222.222 myServerNo2
setComment myServerNo2 Hallo Welt
```

ergibt folgenden Log:

```
Execution of PatchHosts_add
    FILE C:\tmp\testFiles\hosts
  Set ipAddress 111.111.111.111 Hostname "ServerNo1"
  Set Hostname "ServerNo2" for ipAddress 222.222.222.222
  Alias "myServerNo1" set for entry "ServerNo1"
  Alias "myServerNo2" set for entry "222.222.222.222"
  SetComment of Host "myServerNo2" to "Hallo Welt"
  C:\tmp\testFiles\hosts saved back
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_patch_hosts$ = "on"*.

Die Anweisungen im einzelnen:

- `setaddr` <hostname> <ipaddresse>
  Setzt die IP-Adresse für den Host <hostname> auf <ipadresse>. Falls noch kein Eintrag für den Host <hostname> besteht, wird er neu eingetragen.
- `setname` <ipaddresse> <hostname>
  Setzt den Namen des Hosts mit der angegeben IP-Adresse <ipadresse> auf <hostname>. Falls noch kein Eintrag mit der IP-Adresse <ipadresse> existiert, wird er neu erzeugt.
- `setalias` <hostname> <alias>
  Fügt für den Host mit dem IP-Namen <hostname> einen ALIAS-Namen <alias> ein.
- `setalias` <IPadresse> <alias>
  Fügt für den Host mit der IP-Adresse <IPadresse> einen ALIAS-Namen <alias> ein.
- `delalias` <hostname> <alias>
  Löscht aus dem Eintrag für den Host mit dem IP-Namen <hostname> den ALIAS-Namen <alias>.
- `delalias` <IPadresse> <alias>
  Löscht aus dem Eintrag für den Host mit der IP-Adresse <IPadresse> den ALIAS-Namen <alias>.
- `delhost` <hostname> Löscht den Eintrag des Hosts mit dem IP- (oder Alias-) Namen <hostname>.
- `delhost` <IPadresse>
  Löscht den Eintrag des Hosts mit der IP-Adresse <IPadresse>.
- `setComment` <ident> <comment>
  Setzt für den Host mit dem IP-Namen, Alias-Namen oder Adresse <ident> den Kommentareintrag auf <comment>.

8.4 IdapiConfig-Sektionen

Eine IdapiConfig-Sektion waren dazu geeignet, in idapi*.cfg-Dateien, die von der Borland-Database-Engine verwendet werden, die benötigten Parameter einzufügen.

IdapConfig-Sektionen werden vom aktuellen *opsi-winst* nicht mehr unterstützt.

8.5 PatchTextFile-Sektionen

PatchTextFile-Sektionen dienen zum Patchen allgemeiner Textdateien. Es gibt aber auch Spezialanweisungen zum Patchen von Mozilla Konfigurationsdateien.

Wichtig für die Arbeit mit Textdateien ist das Überprüfen, ob eine bestimmte Zeile bereits in einer existierenden Datei vorhanden ist. Für diese Zweck gibt es die boolesche Funktionen `Line_ExistsIn` und `LineBeginning_ExistsIn` (vgl. Kapitel "Boolesche Ausdrücke") zur Verfügung.

8.5.1 Aufrufparameter

Der Name der zu patchenden Datei wird als Parameter übergeben.

8.5.2 Kommandos

Zwei Anweisungen dienen speziell dem komfortablen Patchen von Mozilla-Präferenzdateien:

- `Set_Mozilla_Pref` ("<preference type>", "<preference key>", "<preference value>")
  sorgt dafür, dass in die beim Sektionsaufruf spezifizierten Datei die Zeile <Präferenzvariable> nach <Wert> geschrieben wird. Die ASCII-alphabetische Anordnung der Datei wird beibehalten bzw. hergestellt.
  In den momentanen Mozilla Präferenzdateien gibt es folgende Ausdrücke
  user_pref("<key>", "<value>")
  pref("<key>", "<value>")
  lock_pref("<key>", "<value>")
  Jeder dieser Ausdrücke, sozusagen jede (javascript) Funktion, die auf diese Weise aufgerufen wird
  functionname (String1, String2)
  kann mit diesem Kommando gepatcht werden über das Setzen des entsprechenden Strings für <preference type> (das ist bspw. für *functionname*).
  Wenn der Starteintrag *"functionname (String1)"* in dem bearbeitenden File existiert, kann er gepatcht werden (und bleibt an seinem Platz). Andernfalls wird einen neue Zeile eingefügt.
  Für den *opsi-winst* - ungewöhnlicherweise - sind alle Strings case sensitive.

- `Set_Netscape_User_Pref` ("<Präferenzvariable>", "<Wert>") ist die restriktivere, ältere Version des vorherigen Kommandos und sollte nicht mehr verwendet werden.
  Setzen der Zeile mit den vom User vergebenen Präferenzen für die Variable <Präferenzvariable> und des Wertes <value>.(Abgekündigt!)

- `AddStringListElement_To_Mozilla_Pref` ("<Präferenztyp>", "<Präferenzvariable>", "<add value>")
  fügt ein Element zu einem Listeneintrag der Präferenzdatei hinzu. Das Element wird überprüft, wenn der Wert, der hinzugefügt werden soll, bereits in der Liste existiert (dann wird er nicht hinzugefügt).

- `AddStringListElement_To_Netscape_User_Pref` ("<Präferenzvariable>", "<Werteliste>")
  ist die restriktivere, ältere Version des vorherigen Kommandos und sollte nicht mehr verwendet werden.
  Es fügt einer Werteliste ein Element hinzu (soweit nicht schon enthalten). Angewendet werden kann die Anweisung zur Ergänzung der No-Proxy-Einträge in der prefs.js. (Abgekündigt!)

Alle übrigen Anweisungen von PatchTextFile-Sektionen sind nicht auf spezielle Dateiarten bzw. eine spezielle Syntax der Datei festgelegt:

Die drei Suchanweisungen

- `FindLine` <Suchstring>
- `FindLine_StartingWith` <Suchstring>
- `FindLine_Containing` <Suchstring>

durchsuchen die Datei ab der Position, auf der der Zeilenzeiger steht. Sofern sie eine passende Zeile finden, setzen sie den Zeilenzeiger auf die erste Zeile, die <Suchstring> gleicht / mit ihm beginnt / ihn enthält.

Wird <Suchstring> nicht gefunden, so bleibt der Zeilenzeiger an der Ausgangsposition stehen.

- `GoToTop`
  setzt den Zeilenzeiger vor die erste Zeile setzt (werden Zeilen gezählt muss man berücksichtigen, dass dieses Kommando den Zeilenzeiger über die Anfangszeile setzt). Der Zeilenzeiger kann vor und zurück bewegt werden mit der <Anzahl Zeilen>.

- `AdvanceLine` [<Anzahl Zeilen>]
  bewegt den Zeilenzeiger um <Anzahl Zeilen> vor oder zurück.

- `GoToBottom`
  setzt den Zeilenzeiger auf die letzte Zeile.

- `DeleteTheLine`
  löscht die Zeile auf der der Zeilenzeiger steht, sofern sich dort eine Zeile befindet (wenn der Zeilenzeiger oben platziert ist, wird nichts gelöscht).
- `AddLine_` <Zeile> oder `Add_Line_` <Zeile>
  <Zeile> wird am Schluss der Datei angehängt.
- `InsertLine` <Zeile> oder `Insert_Line_` <Zeile>
  <Zeile> wird an der Stelle eingefügt, an der der Zeilenzeiger steht.
- `AppendLine` <Zeile> oder `Append_Line` <Zeile>
  <Zeile> wird nach der Zeile eingefügt, an der der Zeilenzeiger steht.
- `Append_File` <Dateiname>
  liest die Zeilen der Datei <Dateiname> ein und fügt sie an den Schluss der gerade bearbeiteten Datei an.
- `Subtract_File` <Dateiname>
  entfernt die Anfangszeilen der bearbeiteten Datei, so weit sie mit den Anfangszeilen der Datei <Dateiname> übereinstimmen.
- `SaveToFile` <Dateiname>
  speichert die bearbeitete Datei als <Dateiname>.
- `Sorted`
  bewirkt, dass die Zeilen alphabetisch (nach ASCII) geordnet sind.

8.5.3 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_patch_text_file$ = "on"*

8.6 LinkFolder-Sektionen

Mit LinkFolder-Sektionen werden u.a. die Einträge im Startmenü, die Links auf dem Desktop u.ä. verwaltet.

Zum Beispiel erzeugt folgende Sektion einen Folder (Ordner) namens „acrobat“ im Programme-Folder des allgemeinen Startmenüs (für alle Nutzer gemeinsam).

```
[LinkFolder_Acrobat]
set_basefolder common_programs

set_subfolder "acrobat"
set_link
  name: Acrobat Reader
  target: C:\Programme\adobe\Acrobat\reader\acrord32.exe
  parameters:
  working_dir: C:\Programme\adobe\Acrobat\reader
  icon_file:
  icon_index:
end_link
```

In einer LinkFolder-Sektion muss zuerst bestimmt werden, in welchem virtuellen Systemfolder die nachfolgenden Anweisungen arbeiten sollen. Dafür existiert die Anweisung
`set_basefolder` *<virtueller Systemfolder>*

Virtuelle Systemfolder, die angesprochen werden können, sind:

desktop, sendto, startmenu, startup, programs, desktopdirectory, common_startmenu, common_programs, common_startup, common_desktopdirectory

Die Folder sind virtuell, weil erst durch das Betriebssystem(-Version) bestimmt wird, an welchem physikalischen Ort des Dateisystems sie real existieren. Im zweiten Schritt werden die Subfolder (bzw. Subfolder-Pfade), in denen Links angelegt werden, mit der Anweisung
`set_subfolder` <Folderpath>
bestimmt und zugleich geöffnet. Der Subfolder versteht sich absolut (mit Wurzel im gesetzten virtuellen Systemfolder). Wenn direkt im Systemfolder gearbeitet werden soll, wird dieser mit
`set_subfolder ""` geöffnet.

Im dritten Schritt können die Links gesetzt werden. Der Befehl verwendet eine mehrzeilige Parameterliste. Sie startet mit
`set_link`
Abgeschlossen wird sie durch
`end_link`.

Die Parameterliste insgesamt hat folgendes Format:

`set_link`
`name:` [Linkname]
`target:` <Pfad und Name des Programms>
`parameters:` [Aufrufparameter des Programms]
`working_dir`: [Arbeitsverzeichnis für das Programm]
`icon_file:` [Pfad und Name der Icon-Datei]
`icon_index:` [Position des gewünschten Icons in der Icon-Datei]
`end_link`

Die Angabe eines *target* ist erforderlich. Alle andere Einträge haben Defaultwerte und können leer sein oder entfallen:

- *name* hat als Defaultwert den Programmnamen,
- *parameters* ist, wenn nichts anderes angegeben ist, ein Leerstring,
- *icon_file* ist, wenn nichts anderes angegeben ist, *target* und
- *icon_index* ist per Default 0.

Achtung
Wenn das referenzierte target auf einem, zum Zeitpunkt der Befehlsausführung nicht erreichbaren, Share liegt, werden alle Bestandteile des Pfades auf das Längenschema 8.3 gekürzt.
Workaround:
Manuelles Erzeugen einer korrekten Verknüpfung zu einem Zeitpunkt, in dem das Laufwerk verbunden ist.
Kopieren der korrekten Link-Datei an einen zur Skriptlaufzeit existenten Ort, z.B. C:\Programme.
Diese Datei ist dann das Link-*target*.

- `delete_element` <Linkname>
  wird der angesprochene Link aus dem geöffneten Folder gelöscht.
- `delete_subfolder` <Folderpath>
  löscht den bezeichneten Folder, wobei Folderpath als absolut bezüglich des gesetzten virtuellen Systemfolders zu verstehen ist.

8.6.1 Beispiele

```
set $list2$ = createStringList ('common_startmenu', 'common_programs', 'common_startup', '
    common_desktopdirectory')
for $var$ in $list2$ do LinkFolder_Dummy

[LinkFolder_Dummy]
```

```
set_basefolder $var$
set_subfolder "Dummy"
set_link
        name: Dummy
        target: C:\Programme\PuTTY\putty.exe
        parameters:
        working_dir: C:\Programme\PuTTY
        icon_file:
        icon_index:
end_link
```

Ergibt folgenden Log:

```
Set  $list2$ = createStringList ('common_startmenu', 'common_programs', 'common_startup', '
    common_desktopdirectory')
    retrieving strings from createStringList [switch to loglevel 7 for debugging]
        (string   0)common_startmenu
        (string   1)common_programs
        (string   2)common_startup
        (string   3)common_desktopdirectory

    retrieving strings from $list2$ [switch to loglevel 7 for debugging]
        (string   0)common_startmenu
        (string   1)common_programs
        (string   2)common_startup
        (string   3)common_desktopdirectory


~~~~~~ Looping through:  'common_startmenu', 'common_programs', 'common_startup', '
    common_desktopdirectory'

  Execution of LinkFolder_Dummy
    Base folder is the COMMON STARTMENU folder
    Created "Dummy" in the COMMON STARTMENU folder
      ShellLink "Dummy" created

  Execution of LinkFolder_Dummy
    Base folder is the COMMON PROGRAMS folder
    Created "Dummy" in the COMMON PROGRAMS folder
      ShellLink "Dummy" created

  Execution of LinkFolder_Dummy
    Base folder is the COMMON STARTUP folder
    Created "Dummy" in the COMMON STARTUP folder
      ShellLink "Dummy" created

  Execution of LinkFolder_Dummy
    Base folder is the COMMON DESKTOPDIRECTORY folder
    Created "Dummy" in the COMMON DESKTOPDIRECTORY folder
      ShellLink "Dummy" created

~~~~~~ End Loop
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_link_folder$ = "on"*.

8.7 XMLPatch-Sektionen

Immer häufiger werden Daten aller Art, insbesondere auch Konfigurationsdaten, als XML-Dokument gespeichert.

Der *opsi-winst* bietet XMLPatch-Sektionen an, um XML-Dokumente zu bearbeiten.

Ähnlich wie bei anderen Sektionen (Registry, Patches, LinkFolder) wird dazu zunächst mit bestimmten Befehlen an die Stelle navigiert, an der gearbeitet werden soll und dann dort Detailkommandos ausgeführt.

Das bedeutet, die Aktionen, die *opsi-winst* ausführen kann, gliedern sich in:

- **die Selektion** eines Sets von Elementen des XML-Dokuments, inklusive der Erzeugung nicht vorhandener Elemente,
- **Patch-Aktionen**, die für alle Elemente eines Sets ausgeführt werden sowie
- **die Ausgabe** von Namen und/Attributen der selektierten Elemente für die weitere Verarbeitung.

8.7.1 Aufrufparameter

Der Name der zu patchenden Datei wird als Parameter übergeben.

Beispiel:
`XMLPatch_mozilla_mimetypes $mozillaprofilepath$ + "\mimetypes.rdf"`

8.7.2 Struktur eines XML-Dokuments

Ein XML-Dokument beschreibt die Logik eines „Baums“ (tree), der sich ausgehend von einer „Wurzel“ (root) – passenderweise document root genannt – in die "Äste" (branches) verzweigt. Jede Verzweigungsstelle, wie auch jedes „Astende“, wird als „Knoten“ bezeichnet (englisch node). Die nachgeordneten Knoten eines Knotens heißen auch Kinderknoten ihres Elternknotens.

In XML wird dieser Baum konstruiert durch Elemente. Der Anfang der Beschreibung eines Elements ist mit einem Tag gekennzeichnet (ähnlich wie in der Web-Auszeichnungssprache HTML), d.h. durch einen spezifischen Markierungstext, der durch „<“ und „>“ umrahmt ist. Das Ende der Beschreibung wird wieder durch ein Tag desselben Typnamens gekennzeichnet, jetzt aber durch „</“ und „>“ geklammert. Wenn es keine nachgeordneten Elemente gibt, kann die getrennte Endmarkierung entfallen, stattdessen wird das öffnende Tag mit „/>“ abgeschlossen.

Einen „V“-Baum – mit einer einzigen Verzweigung in zwei Teiläste – könnte man so skizzieren (Wurzel nach oben gedreht):

```
   |      Wurzelknoten
  / \     Knoten 1 auf Ebene 1  bzw.  Knoten 2 auf Ebene 1
 .   .    Implizit vorhandene Endknoten unterhalb von Ebene 1
```

Er würde in XML folgendermaßen dargestellt:

```
<?xml version="1.0"?>
<Wurzelknoten>
    <Knoten_Ebene-1_Nummer-1>
    </Knoten_Ebene-1_Nummer-1>
    <Knoten_Ebene-1_Nummer-2>
    </Knoten_Ebene-1_Nummer-2>
</Wurzelknoten>
```

Die erste Zeile benennt nur die XML-Definition nach der sich das Dokument richtet. Die weiteren Zeilen beschreiben den Baum.

Die insoweit noch nicht komplizierte Struktur wird dadurch verwickelt, dass bis jetzt nur „Hauptknoten“ vorkommen. Ein Hauptknoten definiert ein „Element“ des Baums und ist durch ein Tag gekennzeichnet. Einem solchen Hauptknoten

können – wie bei der Skizze schon angedeutet – „Unterknoten“ und sogar mehrere Arten davon zugeordnet sein. (Befände sich der Baum in der normalen Lage mit Wurzel nach unten, müssten die Unterknoten „Überknoten“ heißen.) Folgende Arten von Unterknoten sind zu berücksichtigen:

- Nachgeordnete Elemente, z.B. könnte der Knoten Nummer 1 sich in Subknoten A bis C verzweigen:

```
<Knoten_Ebene-1_Nummer-1>
    <Knoten_Ebene-2_A>
    </Knoten_Ebene-2_A>
    <Knoten_Ebene-2_B>
    </Knoten_Ebene-2_B>
    <Knoten_Ebene-2_C>
    </Knoten_Ebene-2_c>
</Knoten_Ebene-1_Nummer-1>
```

- Nur wenn es KEINE nachgeordneten Elemente gibt, kann das Element Text enthalten. Dann heißt es, dass dem Element ein Textknoten untergeordnet ist. Beispiel:

```
<Knoten_Ebene-1_Nummer-2>Hallo Welt
</Knoten_Ebene-1_Nummer-2>
```

- Der Zeilenumbruch, der zuvor nur Darstellungsmittel für die XML-Struktur war, zählt dabei jetzt auch als Teil des Textes! Wenn er nicht vorhanden sein soll, muss geschrieben werden

```
<Knoten_Ebene-1_Nummer-2>Hallo Welt</Knoten_Ebene-1_Nummer-2>
```

- Zum Element können außer dem Hauptknoten noch Attribute, sog. Attributknoten gehören. Es könnte z.B. Attribute „Farbe“ oder „Winkel“ geben, die den Knoten 1 in der Ebene 1 näher beschreiben.

```
<Knoten_Ebene-1_Nummer-1 Farbe="grün" Winkel="65">
</Knoten_Ebene-1_Nummer-1>
```

  Eine derartige nähere Beschreibung eines Elements ist mit beiden anderen Arten von Unterknoten vereinbar.

Zur Auswahl einer bestimmten Menge von Elemente könnten im Prinzip alle denkbaren Informationen herangezogen werden, insbesondere

1. die Elementebene (Schachtelungstiefe im Baum),
2. der Name der Elemente, d.h. Name der entsprechenden Hauptknoten, in der Abfolge der durchlaufenen Ebenen (der „XML-Pfad“),
3. die Anzahl, Namen und Werte der zusätzlich gesetzten Attribute,
4. die Reihenfolge der Attribute,
5. die Reihenfolge der Elemente,
6. sonstige „Verwandtschaftsbeziehungen“ der Elemente und
7. Text-(Knoten-)Inhalte von Elementen.

Im *opsi-winst* ist derzeit die Auswahl nach den Gesichtspunkten (1) bis (3) sowie (7) implementiert:

8.7.3 Optionen zur Bestimmung eines Sets von Elementen

Vor jeder weiteren Operation muss das Set von Elementen bzw. von Hauptknoten bestimmt werden, auf die sich die Operation beziehen soll. Das Set wird Schritt für Schritt ermittelt, indem ausgehend von der Dokumentenwurzel Pfade gebildet werden, die jeweils über akzeptierte nachgeordnete Elemente laufen. Die letzten Elemente der Pfade bilden dann das ausgewählte Set.

Der *opsi-winst* Befehl hierfür lautet

- `OpenNodeSet`

Für die Festlegung der akzeptierten Pfade existiert eine ausführliche und eine Kurzsyntax.

Ausführliche Syntax Die ausführliche Syntax für die Beschreibung eines Elemente-Sets bzw. einer Knoten-Menge ist in der folgenden Variante eines Beispiels zu sehen (vgl. Kochbuch, Kapitel "XML-Datei patchen"):

```
openNodeSet
  documentroot
  all_childelements_with:
   elementname:"define"
  all_childelements_with:
    elementname:"handler"
    attribute: extension value="doc"
  all_childelements_with:
    elementname:"application"
end
```

Kurzsyntax Das gleiche Nodeset beschreibt folgende Kurzsyntax (muss in einer Zeile des Skripts untergebracht werden):

```
openNodeSet 'define /handler value="doc"/application /'
```

In dieser Syntax separieren die Schrägstriche die Schritte innerhalb der Baumstruktur, welche in einer Syntax angegeben werden, die ausführlicher als eine eigene Beschreibung ist.

Selektion nach Text-Inhalten (nur ausführliche Syntax) Die ausführliche Syntax erlaubt auch die Selektion nach Text-Inhalten eines Tags:

```
openNodeSet

  documentroot
  all_childelements_with:
  all_childelements_with:
    elementname:"description"
    attribute:"type" value="browser"
    attribute:"name" value="mozilla"
  all_childelements_with:
    elementname:"linkurl"
    text:"http://www.mozilla.org"
end
```

Parametrisierung der Suchstrategie Bei den bislang aufgeführten Beschreibungen eines Elemente-Sets bleiben allerdings eine ganze Reihe von Fragen offen.

- Soll ein Element akzeptiert werden, wenn der Elementname und die aufgeführten Attribute passen, aber weitere Attribute existieren?
- Soll die Beschreibung im Ergebnis eindeutig sein, d.h. genau ein Element liefern? Und wenn doch die Beschreibung des Pfades auf mehrere Elemente passt, muss dann möglicherweise von einer nicht korrekten Konfigurationsdatei ausgegangen werden?

- Soll umgekehrt auf jeden Fall ein passendes Element erzeugt werden, wenn keines existiert?

Zur Regelung dieser Fragen kann die OpenNodeSet-Anweisung parametrisiert werden. Bei den nachfolgend genannten Parametern überdecken „stärkere“ Einstellungen „schwächere“, z.B. ersetzt eine Fehlermeldung eine ansonsten geforderte Warnung. Die angegebenen booleschen Werte sind die Default-Werte:

```
- error_when_no_node_existing false
- warning_when_no_node_existing true
- error_when_nodecount_greater_1 false
- warning_when_nodecount_greater_1 false
- create_when_node_not_existing false
- attributes_strict false
```

Bei Verwendung der Kurzsyntax der OpenNodeSet-Anweisung muss die Parametrisierung vorausgehen und gilt für alle Ebenen des XML-Baumes. In der ausführlichen Syntax kann sie auch direkt nach der OpenNodeSet-Anweisung erfolgen oder für jede Ebene neu gesetzt werden. Sinnvoll kann letzteres vor allem für die Einstellung der Option „create when node not existing“ (Erstellung von Knoten, wenn es keine gibt) sein.

8.7.4 Patch-Aktionen

Auf der mit OpenNodeSet geöffneten bzw. erzeugten Knotenmenge arbeiten nachfolgende Patch-Anweisungen. Es existieren solche:

- zum Setzen und Löschen von Attributen,
- zum Entfernen von Elementen und
- zum Setzen von Text.
- `SetAttribute` "Attributname" value="Attributwert"
  setzt in jedem Element des aktuellen Knoten- bzw. Elementsets das Attribut auf den genannten Wert. Wenn das Attribut nicht vorhanden ist wird es erzeugt.
  Beispiel:
  `SetAttribute "name" value="OpenOffice Writer"`
- `AddAttribute` "Attributname" value="Attributwert"
  setzt das Attribut dagegen nur auf Attributwert, wenn es vorher nicht existiert, ein vorhandenes Attribut behält seinen Wert. Z.B. würde die Anweisung
  `AddAttribute "name" value="OpenOffice Writer"`
  eine vorher vorhandene Festlegung auf ein anderes Programm nicht überschreiben.
- `DeleteAttribute` "Attributname"
  wird das betreffende Attribut von jedem Element der aktuellen Knotenmenge entfernt.
- `DeleteElement` "Elementname"
  entfernt das Element, dessen Hauptknoten den (Tag-) Namen "Elementname" hat, samt Unterknoten aus der aktuellen Knoten- oder Elementmenge.

Schließlich existieren zwei Anweisungen zum Setzen bzw. Hinzufügen von Textinhalten eines Elements. Die beiden Anweisungen lauten

- `SetText` "Text"

und

- `AddText` "Text"

Z.B. wird, wenn das betreffende Element in der geöffneten Elementmenge liegt, durch die Anweisung
`SetText "rtf"`
aus
<fileExtensions>doc<fileExtensions>
das Element
<fileExtensions>rtf<fileExtensions>

Mit
`SetText ""`
wird der Text komplett entfernt.

`AddText "rtf"`
setzt analog wie bei anderen Add-Anweisungen den Text, sofern kein Text vorhanden ist - existierender Text bleibt unberührt.

8.7.5 Rückgaben an das aufrufende Programm

Eine XMLPatch-Sektion kann angewiesen werden, String-Listen an das rufende Programm zurückzugeben.

Dazu muss sie in einer primären Sektion mit der String-Listen-Anweisung `getReturnListFromSection` aufgerufen werden. Die Anweisung kann in einem String-Listen-Ausdruck verwendet werden, z.B. das Ergebnis einer String-Listen-Variable zugewiesen werden. So kann in der XMLPatch_mime-Sektion stehen:

```
DefStringList $list1$
set $list1$=getReturnListFromSection ('XMLPatch_mime "c:\mimetypes.rdf"')
```

Eine Return-Anweisung in der XMLPatch-Sektion regelt, welche Zeilen die XMLPatch-Sektion als Inhalt der String-Liste ermittelt:

- `return elements`+ Bewirkt, dass die ausgewählten Elemente komplett (Elementname und Attribute) ausgegeben werden.
- `return attributes`
  Erzeugt eine Liste der Attribute.
- `return elementnames`
  Listet die Elementnamen.
- `return attributenames` Produziert eine Liste der Attributnamen.
- `return text`
  Listet die textlichen Inhalte der selektierten Elemente.
- `return counting`
  Liefert eine Listenstruktur mit summarischen Informationen: In Zeile 0 steht die Anzahl aller ausgewählten Elemente, in Zeile 1 die Zahl aller Attribute.

8.7.6 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_xml$ = "on"*

8.8 ProgmanGroups-Sektionen

Dieser Sektionstyp ist abgekündigt.

8.9 WinBatch-Sektionen

In einer WinBatch-Sektion kann jedes Windows-Programm als Anweisung verwendet werden.

Z.B kann mit folgender WinBatch-Sektion ein Setup-Programm gestartet werden:

```
[winbatch_install]
"%scriptpath%\setup.exe"
```

Es ist abgekündigt (aber noch unterstützt) wie auch aus dem Windows Explorer heraus Daten-Dateien, die mit einem Programm verknüpft sind, direkt aufzurufen. Wenn Sie das tun bekommen Sie eine deprecated Warnung.

8.9.1 Aufrufparameter (Modifier)

Durch die Parameter des WinBatch-Aufrufs wird festgelegt, wie sich *opsi-winst* gegenüber den in der WinBatch-Sektion gestarteten Programmen verhält.

- `/WaitOnClose`
  Default
  opsi-winst wartet die Selbstbeendigung des angestoßenen Prozesses ab. Dieses Verhalten kann mit dem Parameter auch explizit definiert werden.
- `/LetThemGo`
  Verschiebt den aufgerufenen Prozess in den Hintergrund und wartet **nicht** auf dessen Beendigung; d.h. das sofort die nächste Zeile der WinBatch-Sektion bzw. die nächste Zeile des übergeordneten Programms abgearbeitet werden.
- `/WaitSeconds` [AnzahlSekunden]
  Die Parametrisierung /WaitSeconds [AnzahlSekunden] modifiziert das Verhalten dahingehend, dass *opsi-winst* jeweils erst nach [AnzahlSekunden] die Skriptbearbeitung fortsetzt. Die angegebene Zeit stoppt *opsi-winst* auf jeden Fall. In der Default-Einstellung wird zusätzlich auf das Ende der angestoßenen Prozesse gewartet. Ist letzteres nicht gewünscht, so kann der Parameter mit dem Parameter /LetThemGo kombiniert werden.
- `/WaitForWindowAppearing` [Fenstertitel]
  bzw.
  `/WaitForWindowVanish` [Fenstertitel]
  Abgekündigt. Verwenden Sie `/WaitForProcessEnding`
  Im 1. Fall wartet *opsi-winst* solange, bis ein Prozess, der sich durch ein mit [Fenstertitel] benanntes Fenster kenntlich macht, gestartet ist. Im 2. Fall wartet *opsi-winst* bis ein, mit [Fenstertitel] benanntes, Fenster auf dem Desktop erst einmal erscheint und dann auch wieder geschlossen wird. Auf diese Weise kann unter geeigneten Umständen geprüft werden, ob sekundäre, indirekt gestartete Prozesse sich beendet haben.

  

  Achtung
  Diese Befehle erkennen nur Fenster von 32 Bit-Programmen.

- `/WaitForProcessEnding` <program name>
  Wartet darauf, das sich der Prozess mit dem Namen <program name> beendet.
  Kann und sollte mit `/TimeOutSeconds` kombiniert werden.

Erläuterung:
Der *opsi-winst* wartet auf das Ende eines per `winbatch` gestarteten Prozesses bevor mit der nächsten Zeile des Scriptes forgefahren wird:

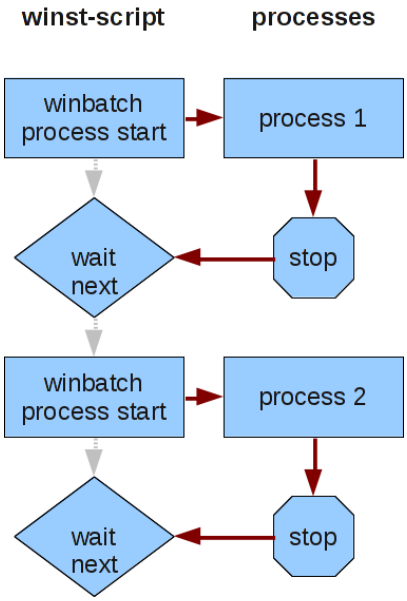

Abbildung 8.1: Sequentielle Abarbeitung des Scriptes mit Warten auf das Ende eines Prozesses

Es gibt allerdings Prozesse, welche einen weiteren Prozess starten und sich Beenden ohne auf das Ende des Kindprozesses zu warten. Aus Sich des *opsi-winst* ist admit der Weg zur Ausführung des nächsten Befehls frei:

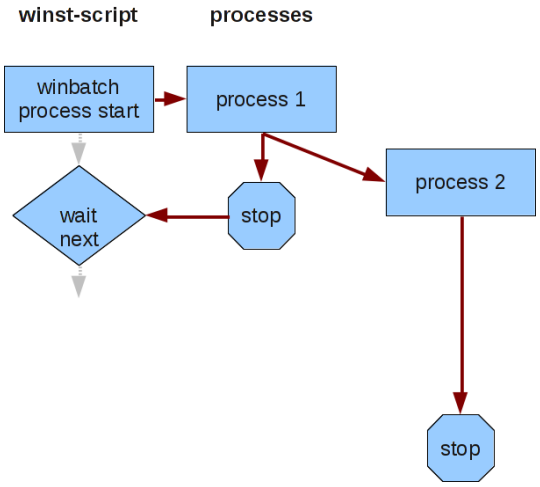

Abbildung 8.2: Ende eines Prozesses mit weiterlaufenden Kindprozess

Werden z.B. hintereinander ein Uninstall und ein Setup Programm aufgerufen und das Uninstall Programm führt die eigentliche Deinstallation in einem Kindprozess aus, so ist das Ergebnis undefiniert, da deinstallation und Installation gleichzeitig laufen:

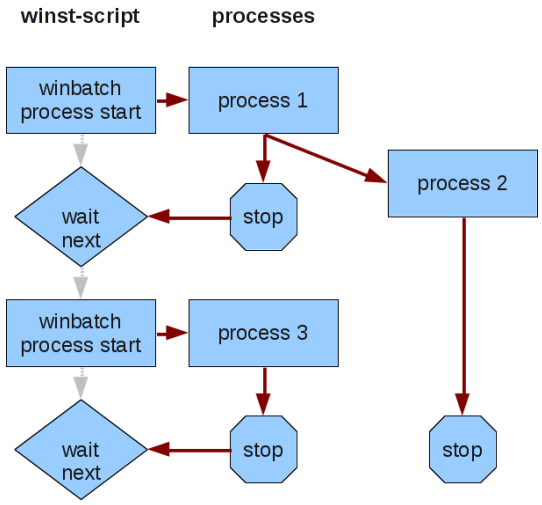

Abbildung 8.3: Überlappung von Kindprozess und nächstem gestarteten Prozess

Mit dem Modifier `/WaitForProcessEnding` kann eine solche Situation vermieden werden.

- `/TimeOutSeconds` <seconds>
  Bricht das Warten auf das Processende oder eine Wartebedingung (`/WaitForProcessEnding`) nach Ablauf von <seconds> ab, auch wenn das Prozessende oder die Wartebedingung noch nicht erfüllt ist.
  Der Prozess auf dessen Ende gewartet werden sollte wird nicht gestopt.
  Kann seit Version 4.11.3 auch alleine (z.B. ohne `/WaitForProcessEnding`) verwendet werden, aber nicht zusammen mit `/WaitSeconds`.
  Beispiel:

```
Winbatch_uninstall /WaitForProcessEnding "uninstall.exe" /TimeOutSeconds 20
[Winbatch_uninstall]
"%ScriptPath%\uninstall_starter.exe"
```

- `/RunElevated`
  Started dem Prozeß mit einem höheren Security Token (d.h. mit höheren Rechten). Dieser Modifier hat folgende Einschränkungen:

  - Unter NT5 hat er keine Auswirkungen
  - Ein Zugriff auf das Netz ist in dem Prozess nicht möglich. Daher müssen die aufzurufenden Programme von einem Netzlaufwerk in ein temporäres lokales Verzeichnis kopiert werden.
  - Evtl. kann es zu Problemen bei der Nutzung der grafischen Oberfläche kommen. Daher sind echte silent aufrufe hier zu bevorzugen.
  - Funktioniert nur im opsi-service Kontext

- `getLastExitCode`
  Die String-Funktion getLastExitCode gibt den ExitCode des letzten Prozessaufrufs der vorausgehenden WinBatch / DosBatch / ExecWith Sektion aus.

8.9.2 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_winbatch$ = "on"*

8.10 DOSBatch/DosInAnIcon (ShellBatch/ShellInAnIcon) Sektionen

DOSBatch-Sektionen (auch ShellBatch genannt) sollen in erster Linie dazu dienen, vorhandene Kommandozeilenroutinen für bestimmte Zwecke zu nutzen. *opsi-winst* wartet auf die Beendigung des DOS-Batch, bevor die nächste Sektion des Skripts abgearbeitet wird.

Eine DosBatch-Sektion wird bei der Abarbeitung des Skripts in eine temporäre Batch-Datei winst **.bat* umgewandelt. Da die Datei in *c:\tmp* angelegt wird, muss dieses Verzeichnis existieren und zugänglich sein. Die Batch-Datei wird dann in einem Kommando-Fenster mit cmd.exe als Kommando-Interpreter ausgeführt. Dies erklärt warum in einer DosBatch Sektion alle Windows Shell Kommandos verwendet werden können.

Gegenüber dem Aufruf einer cmd-Datei per Winbatch-Sektion bietet die DosBatch Sektionen drei Vorteile:

- In der Sektion vorhandene *opsi-winst* Variablen oder Konstanten werden vor der Ausführung durch Ihren Inhalt ersetzt und können so unkompliziert verwendet werden.
- Die Ausgaben des Aufrufs werden in der Logdatei abgespeichert.
- Die Ausgaben des Aufrufs können einer String-Liste übergeben und weiterverarbeitet werden.

Der Sektionstyp DOSInAnIcon oder ShellInAnIcon ist identisch mit der betreffenden DOSBatch Syntax und ausführenden Methoden. Allerdings wird das aufgerufenen Fenster minimiert dargestellt.

Achtung
Verwenden Sie keine Kommandos, die auf Eingaben warten.

8.10.1 Aufrufparameter

Zu unterscheiden ist zwischen Parametern die der aufgerufenen Batchdatei übergeben werden und denen die *opsi-winst*-intern verwendet werden. Der Aufrufsyntax ist daher:

```
Sektionsname [batch parameter] [winst [modifier]]
```

Erlaubte winst modifier sind (seit 4.11.1):

- `/32bit`
- `/64bit`
- `/Sysnative`

Parameter des Aufrufs der DosBatch-Sektion in der Actions-Sektion werden unmittelbar als Parameter der Batch-Datei interpretiert.

Zum Beispiel bewirken die Anweisungen in Actions-Sektionen bzw. der Sektion DosBatch_1 :

```
[Actions]
DosBatch_1 today we say "Hello World"

[DosBatch_1]
@echo off
echo %1 %2 %3 %4
pause
```

die Ausführung des Dos-Batch-Befehls echo mit Parametern *today we say "Hello World"*.

Das folgende Beispiel wird auf einem 64 Bit System mit einer 64 Bit cmd.exe gestartet und erzeugt die Ausgabe *today we say*:

```
[Actions]
DosBatch_1 today we say winst /64bit

[DosBatch_1]
@echo off
echo %1 %2 %3 %4
pause
```

8.10.2 Einfangen der Ausgaben

Sollen die Ausgaben, die von Befehlen einer DosBatch-Sektion kommen, aufgefangen werden, so geschieht dies mittels `getOutStreamFromSection ()` aus der Haupt-Sektion des opsi-winst-Skripts (siehe Kapitel "(Wieder-) Gewinnen von Einzelstrings aus String-Listen").

Sollen die zurückgegebenen Strings weiterverarbeitet werden, so wird dringend geraten, vor den Befehlszeilen ein *@*-Zeichen zu verwenden bzw. die Kommandos mit *@echo off* zu beginnen. Dies unterdrückt die Ausgabe der Befehlszeile selbst, die je nach System anders formatiert sein kann.

8.10.3 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_dos$ = "on"*

8.11 Registry-Sektionen

Diese Funktion ist nur unter Windows verfügbar.

Registry-Sektionen dienen dem Erzeugen und Patchen von Einträgen in der Windows-Registrierdatenbank, wobei die Eintragungen mit dem opsi-winst-üblichen Detaillierungsgrad protokolliert werden.

8.11.1 Beispiele

Man kann eine Registry-Variable setzen indem man die Sektion mit Registry_TestPatch aufruft, wo sie dann wie folgt angegeben ist

```
[Registry_TestPatch]
openkey [HKEY_Current_User\Environment\Test]
set "Testvar1"  = "c:\rutils;%Systemroot%\hey"
set "Testvar2" = REG_DWORD:0001
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_subregistry$ = "on"*

8.11.2 Aufrufparameter

- Die Standardform der Registry-Sektionen ist unparametrisiert. Dies genügt, weil auf dem Windows-PC nur eine einzige Registrierdatenbank gibt und somit das globale Ziel der Bearbeitung feststeht.
- `/AllNTUserDats`
  Es gibt jedoch die Möglichkeit, dass die Patches einer Registry-Sektion automatisch für "alle NT User", entsprechend den verschiedenen User-Zweigen der Registry, vorgenommen werden. Das entsprechende Verfahren bei der Abarbeitung der Sektion wird mit dem Parameter `/AllNTUserDats` aufgerufen.

Außerdem kontrollieren Parameter mit welchen syntaktische Varianten Registry-Sektionen angefordert werden kann:

- `/regedit`
  Wird das Registry-Kommando mit dem Parameter `/regedit` verwendet, so kann der Export eines Registry-Teilzweiges mit dem Programm, der mit dem gewöhnlichen Windows-Registry-Editor regedit erstellt wurde, direkt als Eingabedatei für Registry dienen (vgl. Abschnitt "Registry-Sektionen im Regedit-Format").

- `/addReg`
  Eine weitere Variante des Registry-Aufrufs dient dazu, die Patch-Anweisungen für die Registry zu verarbeiten, die im inf-Datei-Standard erstellt sind. Zur Kennzeichnung dient der Parameter /addReg (in Anlehnung an die entsprechende Abschnittsbezeichnung in einer inf-Datei)(vgl. Abschnitt "Registry-Sektionen im AddReg-Format").

Diese nicht *opsi-winst* spezifischen syntaktischen Varianten sind im Handbuch nicht beschrieben, da sie normalerweise automatisch generiert werden.

Weiterhin gibt es die Aufrufparameter,

- `/32Bit`
- `/64Bit`
- `/SysNative`

welche auf 64 Bit-Systemen das Schreiben in den 32 Bit- bzw. 64 Bit-Zweig der Registry beeinflusst (siehe Kapitel 64 Bit-Unterstützung).

8.11.3 Kommandos

Die Syntax der Defaultform einer Registry-Sektion ist an der Kommandosyntax anderer Patchoperationen des *opsi-winst* orientiert.

Es existieren die Anweisungen:

- `OpenKey`
- `Set`
- `Add`
- `Supp`
- `GetMultiSZFromFile`
- `SaveValueToFile`
- `DeleteVar`
- `DeleteKey`
- `ReconstructFrom`
- `Flushkey`

Im Detail:

- `OpenKey` <Registryschlüssel>
  Öffnet den bezeichneten Schlüssel in der Registry zum Lesen (und wenn der eingeloggte User über die erforderlichen Rechte verfügt zum Schreiben); existiert der Schlüssel noch nicht, wird er erzeugt.

Registry-Schlüssel sind ja hierarchisch organisierte Einträge Registrierungsdatenbank. Die hierarchische Organisation drückt sich in der mehrstufigen Benennung aus: Für die oberste (Root-) Ebene können standardmäßig insbesondere die "high keys" *HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS* und *HKEY_CURRENT_CONFIG* verwendet werden. Gebräuchliche Abkürzungen sind *HKCR, HKCU, HKLM* und *HKU*.

In der *opsi-winst* Syntax bei den Registry-Pfaden werden die weiteren folgenden Ebenen jeweils durch einen Backslash getrennt.

Alle anderen Kommandos arbeiten mit einem geöffneten Registry-Key.

- `Set` <Varname> = <Value>
  setzt die durch <Varname> bezeichnete Registry-Variable auf den Wert <Value>, wobei es sich sowohl bei <Varname> als auch bei <Value> um Strings handelt, die in Anführungszeichen eingeschlossen sind. Existiert die Variable noch nicht, wird sie erzeugt. Dabei wird als Default Datentyp *REG_SZ* verwendet. Enthält allerdings <value> ein oder mehrere Prozentzeichen (%) so wird als Datentyp *REG_EXPAND_SZ* verwendet.

Es gibt auch den Leerstring als Variablenname; dieser entspricht dem "(Standard)"-Eintrag im Registry-Schlüssel.

Soll eine Registry-Variable erzeugt oder gesetzt werden, bei der der Datentyp explizit angegeben werden soll, muss die erweiterte Form der Set-Anweisung verwendet werden:

- `Set` <Varname> = <Registrytyp>:<Value>
  Setzt die durch <Varname> bezeichnete Registry-Variable auf den Wert <Value> des Typs <Registrytyp>. Es werden folgende Registry-Typen interpretiert:

  REG_SZ
  (String)

  REG_EXPAND_SZ
  (ein String, der vom System zu expandierende Teilstrings wie %Systemroot% enthält)

  REG_DWORD
  (ganzzahlige Werte)

  REG_BINARY
  (binäre Werte, in zweistelligen Hexadezimalen, d.h. 00 01 02 .. 0F 10 .., notiert)

  REG_MULTI_SZ
  (Arrays von String-Werten, die in opsi-winst-Syntax durch das Zeichen "|" getrennt werden;
  Beispiel für REG_MULTI_SZ:

```
set "myVariable" = REG_MULTI_SZ:"A|BC|de"
```

Wenn ein Multi-String zunächst zusammengestellt werden soll, kann dies zeilenweise in einer Datei geschehen, die man dann mit Hilfe der Anweisung GetMultiSZFromFile (s.u.) einliest.

- `Add` <Varname> = <Value>

  bzw.

  `Add` <Varname> = <Registrytyp> <Value>
  arbeitet analog zu Set mit dem Unterschied, dass nur Variablen hinzugefügt, Einträge für bestehende Variablen nicht verändert werden.

- `Supp` <Varname> <Listenzeichen> <Supplement>
  Dieses Kommando liest den String-Wert der Variablen <varname>, einer Liste aus Werten, die separiert werden durch <Listenzeichen> und den String <supplement> zu dieser Liste (wenn sie noch nicht enthalten sind), aus. Wenn <supplement> die <separator> enthält, können mit diesen Listenzeichen die Einträge in einzelne Strings unterteilt werden und die Prozedur wird für jeden Teilstring angewendet.

  Eine typische Verwendung ist der Eintrag zu einer Pfadvariablen, die in der Registry definiert ist.

  Supp behält den ursprünglichen Stringtyp (REG_EXPAND_SZ bzw. REG_SZ) bei.

Beispiel:

Der allgemeine Systempfad wird festgelegt durch den Eintrag der Variable Path im Registrierschlüssel

KEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment

Wenn dieser Schlüssel mit OpenKey geöffnet ist, kann mit der Anweisung

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

der Pfad ergänzt werden, um die Einträge *"C:\utils"* sowie *"%JAVABIN%"*.

(Weil der Registry-Eintrag für den Systempfad den Datentyp REG_EXPAND_SZ hat, expandiert Windows %JAVABIN% automatisch zum entsprechenden Verzeichnisnamen, falls %JAVABIN% ebenfalls als Variable definiert ist).

Unter Win2k ist das Phänomen zu beobachten, dass sich der path-Eintrag nur per Skript auslesen (und dann patchen) lässt, wenn vor dem Lesen ein Wert gesetzt wird.

Der alten Wert von Path wird aus der Umgebungsvariable auslesen, wieder in die Registry zurückgeschrieben und dann ist es möglich mit der Registry-Variablen zu arbeiten.

```
[Actions]
DefVar $Path$
set $Path$ = EnvVar ("Path")
Registry_PathPatch

[Registry_PathPatch]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\control\Session Manager\Environment]
set "Path"="$Path$"
supp "Path"; "c:\orawin\bin"
```

Achtung
Nach dem Patchen des Registry-Path enthält die Umgebungsvariable Path den veränderten Wert erst nach einem Reboot.

- `GetMultiSZFromFile` <varname> <Dateiname>
  Liest eine Datei zeilenweise in einen Multistring ein und weist diesen <varname> zu.
- `SaveValueToFile` <varname> <filename>
  Exportiert die genannten Werte (String oder MultiSZ) in die Datei <filename>
- `DeleteVar` <Varname>
  Löscht den Eintrag mit Bezeichnung <Varname> aus dem geöffneten Schlüssel.
- `DeleteKey` <Registryschlüssel>
  Löscht den Registry-Key rekursiv samt aller Unterschlüssel und den enthaltenen Registry-Variablen und -Werten. Zur Syntax, in der der Registrierschlüssel angegeben wird, vgl. OpenKey.

  Beispiel:

  ```
  [Registry_KeyLoeschen]
  deletekey [HKCU\Environment\subkey1]
  ```

- `ReconstructFrom` <Dateiname>
  (abgekündigt)
- `FlushKey`
  Sorgt dafür, dass die Einträge des Schlüssels nicht mehr nur im Speicher gehalten, sondern auf die Platte gespeichert werden (geschieht automatisch beim Schließen eines Keys, insbesondere beim Verlassen einer Registry-Sektion).

8.11.4 Registry-Sektionen, die `alle NTUser.dat` patchen

Wird eine Registry-Sektion mit dem Parameter `/AllNTUserdats` aufgerufen, so werden ihre Anweisungen für alle auf dem NT-System angelegten User ausgeführt.

Dazu werden zunächst die Dateien NTUser.dat für alle auf dem System eingerichteten User-Accounts durchgegangen (in denen die Registry-Einstellungen aus *HKEY_Users* abgelegt sind). Sie werden temporär in einen Hilfszweig der Registry geladen und dort entsprechenden der Anweisungen der Sektion bearbeitet. Weil dies für den zum Zeitpunkt der Programmausführung angemeldeten User nicht funktioniert, werden die Anweisungen der Sektion zusätzlich für *HKEY_Current_User* ausgeführt. Als Ergebnis verändert sich die gespeicherte NTUser.dat.

Dieser Mechanismus funktioniert nicht für einen angemeldeten User, da seine NTUser.dat in Benutzung ist und der Versuch die Datei zu laden einen Fehler produziert. Damit aber auch für den angemeldeten User Änderungen durchgeführt werden, werden die Registry Kommandos ebenfalls auf den Bereich *HKEY_Current_User* angewendet (*HKEY_Users* ist der Zweig für den angemeldeten Benutzer).

Auch künftig erst angelegte Accounts werden mit erfasst, da auch die NTUser.dat aus dem Profilverzeichnis des *Default Users* bearbeitet wird.

Die Syntax der Sektion ist die einer Standard-Registry-Sektion. Allerdings werden bis vor Version 4.11.2.1 alle Schlüsselnamen relativ interpretiert. D.h. **der Hauptkey ist wegzulassen**: Im folgenden Beispiel werden faktisch die Registry-Einträge für die Variable *FileTransferEnabled* unter *HKEY_Users\XX\Software...* neu hergestellt, sukzessive für alle User auf der Maschine:

```
[Registry_AllUsers]
openkey [Software\ORL\WinVNC3]
set "FileTransferEnabled"=reg_dword:0x00000000
```

Seit *opsi-winst* version 4.11.2 darf man den root key *HKEY_CURRENT_USER* beim `openkey` Kommando mitgeben. Beispiel:

```
[Registry_AllUsers]
openkey [HKEY_CURRENT_USER\Software\ORL\WinVNC3]
set "FileTransferEnabled"=reg_dword:0x00000000
```

Das hat folgende Vorteile:

- Der Syntax ist leichter verständlich
- Die selbe Registry Sektion kan mit */AllNtuserdats* und in einem *userLoginScript* verwendet werden.

8.11.5 Registry-Sektionen im Regedit-Format

Bei Aufruf von Registry mit dem Parameter `/regedit` wird der Inhalt der Registry-Sektion in dem Exportformat erwartet, dass das Standard-Windows-Programm regedit erzeugt.

Die von regedit generierten Exportdateien haben – von der Kopfzeile abgesehen - den Aufbau von Ini-Dateien haben. Beispiel:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org]

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\general]
"bootmode"="BKSTD"
"windomain"=""
"opsiconf"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\shareinfo]
"user"="pcpatch"
"pcpatchpass"=""
"depoturl"="\\\\bonifax\\opt_pcbin\\install"
```

```
"configurl"="\\\\bonifax\\opt_pcbin\\pcpatch"
"utilsurl"="\\\\bonifax\\opt_pcbin\\utils"
"utilsdrive"="p:"
"configdrive"="p:"
"depotdrive"="p:"
```

Die Sektionen bezeichnen hier Registry-Schlüssel, die geöffnet werden sollen. Die einzelnen Zeilen stehen für die gewünschten Setzungen von Variablen (entsprechend dem Set-Befehl in opsi-winst-Registry-Sektionen).

Diese Anweisungen können aber nun nicht als Sektion innerhalb eine *opsi-winst* Skripts untergebracht werden. Daher kann die Registry Sektion mit dem Parameter `/regedit` nur als ausgelagerte Sektion oder über die Funktion loadTextFile geladen werden:

```
registry "%scriptpath%/opsiorgkey.reg" /regedit
```

Zu beachten ist noch, dass regedit ab Windows XP nicht mehr das Regedit4-Format produziert, sondern ein Format, dass durch die erste Zeile
"Windows Registry Editor Version 5.00"
gekennzeichnet ist.

Windows sieht hier zusätzliche Wertetypen vor. Gravierender ist, dass die Exportdatei ursprünglich in Unicode erzeugt wird. Um sie mit den 8 Bit-Mitteln der Standardumgebung des *opsi-winst* zu verarbeiten, muss der Zeichensatz konvertiert werden. Die Konvertierung kann z.B. mit einem geeigneten Editor durchgeführt werden. Eine andere Möglichkeit besteht darin, die Konvertierung on the fly vom *opsi-winst* durchführen zu lassen. Dazu lässt sich die String-Listenfunktion `loadUnicodeTextFile` verwenden. Wenn z.B. printerconnections.reg ein Unicode-Export ist, wäre regedit in folgender Form aufzurufen:

```
registry loadUnicodeTextFile("%scriptpath%/opsiorgkey.reg") /regedit
```

Auch eine Registry-Patch im regedit-Format kann „für alle NT-User“ ausgeführt werden, sinngemäß in der gleichen Weise wie oben für das gewöhnliche winst-Registry-Patch-Format beschrieben. D.h. der Root-Schlüssel *HKCU* muss aus den Angaben entfernt werden und dann wird aus + *[HKEY_CURRENT_USER\Software\ORL]* → *[Software\ORL]*.

8.11.6 Registry-Sektionen im AddReg-Format

Die Syntax einer Registry-Sektion, die mit dem Parameter `/addReg` aufgerufen wird, folgt der Syntax von *[AddReg]*-Sektionen in inf-Dateien, wie sie z.B. von Treiberinstallationen verwendet wird.

Beispiel:

```
[Registry_ForAcroread]
HKCR,".fdf","",0,"AcroExch.FDFDoc"
HKCR,".pdf","",0,"AcroExch.Document"HKCR,"PDF.PdfCtrl.1","",0,"Acr"
```

8.12 OpsiServiceCall Sektion

Mit dieser Sektion ist es möglich Informationen abzufragen – oder Daten zu bestimmen – mit Hilfe des opsi Service. Es gibt drei Optionen, mit denen man die Verbindung zum opsi Service definieren kann:

- Per Voreinstellung wird vorausgesetzt, dass das Skript in der Standard opsi Installationsumgebung ausgeführt werden kann. D.h. es besteht eine Verbindung zum opsi Service, die genutzt wird.
- Es wird eine URL für den gewünschten Service und ebenso der benötigte Benutzername und das Passwort als Sektionsparameter gesetzt.
- Es kann ein interaktives Login für den Service gesetzt werden – mit einer voreingestellten Service URL und dem Benutzernamen, wenn das gewünscht wird.

Die abgerufenen Daten können als String-Liste zurückgegeben und dann für die Verwendung in Skripten benutzt werden.

8.12.1 Aufrufparameter

Es gibt Optionen, mit denen man die Verbindung zu einem opsi Service angeben kann und Einstellungen, die für die Verbindung benötigt werden.

Verbindungsparameter können mit Hilfe von

- `/serviceurl` <url to the opsi web service>
- `/username` <web service user name>
- `/password` <web service user password>

gesetzt werden. Wenn diese Parameter definiert sind (oder zumindest einer der Parametern), wird versucht eine Verbindung zu der genannten Service URL herzustellen.

Wenn kein Parameter ausgewiesen ist, kann eine bestehende Verbindung wieder verwendet werden.

Mit der Option

- `/interactive`
  kann man bestimmen, dass eine interaktive Verbindung benutzt werden soll. Das bedeutet, dass der Benutzer die Verbindungsdaten bestätigen muss und das Passwort eingibt. Diese Option kann damit nicht in Skripten verwendet werden, die voll automatisch ausgeführt werden sollen.

Wenn keine Verbindungsparameter vorgegeben sind setzt der *opsi-winst* vorraus, dass die bestehende Verbindung wieder benutzt werden soll.

Wenn keine Verbindungsparameter angegeben und auch die Option „interactive“ nicht ausgewählt wird (weder bei diesem Skript noch zuvor), wird vorausgesetzt das der *opsi-winst* mit dem Standard opsi Bootprozess arbeitet und sich darüber mit dem opsi Service verbindet. Theoretisch gibt es eine Verbindung zu einem zweiten opsi Service, die als Verbindung zu dem Standard opsi Service mit der Option

- `/preloginservice`
  neu gestartet werden kann.
- `/opsiclientd` //since 4.11.2.1
  ruft den Webservice des lokalen opsiclientd.

8.12.2 Sektionsformat

Ein `opsiServiceCall`, welcher eine existierende Verbindung zu einem opsi Service benutzt, wird bestimmt durch den Methodennamen und eine Parameterliste.

Beide werden in dem Sektionsabschnitt definiert und haben folgendes Format:

```
"method":<method name>
"params":[
        <params>
        ]
```

Dabei sind *<params>* kein, ein oder auch mehrere durch Komma getrennte Strings. Welche Parameter benötigt werden, hängt von der aufgerufenen Methode ab.

Beispiel:

```
[opsiservicecall_clientIdsList]
"method":"getClientIds_list"
"params":[]
```

Die Sektion erstellt eine Liste der PC-Namen (IDs) von allen lokalen opsi Benutzern. Wenn es für andere Zwecke als Test und Dokumentation genutzt werden soll, kann die Sektion als ein Teil eines String-Listen Ausdrucks (vgl. das folgende Beispiel) verwendet werden.

```
DefStringList $result$
Set $result$=getReturnListFromSection("opsiservicecall_clientIdsList")
```

Die Verwendung von GetReturnListFromSection ist dokumentiert in dem Kapitel zur String-Listenverarbeitung dieses Handbuchs (siehe Kapitel "String-Listen-Erzeugung mit Hilfe von Sektionsaufrufen").

Ein Hash, der eine Namensliste mit Wertepaaren enthält, wird durch den folgenden opsi Service aufgerufen (beinhaltet keine leere Parameterliste):

```
[opsiservicecall_hostHash]
"method": "getHost_hash"
"params": [
        "pcbon8.uib.local"
        ]
```

Achtung
Die opsiservicecall Sektionen sind für opsi 3.x Methoden entwickelt worden und für die opsi 4.x Methoden häufig nicht verwendbar. So sind **_getIdents* Aufrufe zwar möglich, **_getObjects* Aufrufe aber nicht.

8.12.3 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_opsiServiceCall$ = "on"*

8.13 ExecPython Sektionen

Die ExecPython Sektionen basieren auf Shell-Sektionen (ähnlich wie DosInAnIcon). Während diese den Inhalt der Sektion dem Interpreter cmd.exe übergeben, wird der Inhalt einer ExecPython Sektion dem Python Interpreter übergeben (welcher auf dem System installiert sein muss).

Beispiel

Das folgende Beispiel demonstriert einen execPython Aufruf mit einer Parameterliste zu dem *print* Python-Kommando.

Der Aufruf könnte wie folgt aussehen

```
execpython_hello -a "option a" -b "option b" "there we are"

[execpython_hello]
import sys
print "we are working in path: ", a
if len(sys.argv) > 1 :
        for arg in sys.argv[1:] :
                print arg
else:
  print "no arguments"


print "hello"
```

Die Ausgabe des Druck-(print) Kommandos wird gesammelt und in einen Logdatei geschrieben. So kann man die folgende Logdatei bekommen

```
output:
 --------------
-a
option a
-b
option b
there we are
       hello
```

Anzumerken ist hierbei, dass der loglevel auf *1* gesetzt werden muss, damit die Ausgabe wirklich den Weg in die Logdatei findet.

8.13.1 Verflechten eines Python Skripts mit einem *opsi-winst* Skript

Aktuell ist die execPython Sektion dem *opsi-winst* Skript über vier Kategorien von gemeinsam genutzten Daten integriert:

- Eine Parameterliste geht zum Python Skript über.
- Alles was vom Python Skript gedruckt wird, wird in die *opsi-winst* log-Datei geschrieben.
- Der *opsi-winst* Skript Mechanismus für die Einführung von Konstanten und Variablen in Sektionen arbeitet erwartungsgemäß für die execPython Sektion.
- Die Ausgabe einer execPython Sektion kann umgewandelt werden in eine String-Liste und dann vom laufenden *opsi-winst* Skript weiter verwendet werden.

Ein Beispiel für die ersten beiden Wege der Verflechtung des Python Skripts mit dem *opsi-winst* Skript werden im Anschluss beschrieben. Es wurde erweitert, damit einige der Werte von *opsi-winst* Konstanten oder Variablen aufgerufen werden können.

```
[execpython_hello]
import sys
a = "%scriptpath%"
print "we are working in path: ", a
print "my host ID is ", "%hostID%"
if len(sys.argv) > 1 :
        for arg in sys.argv[1:] :
                print arg
else:
  print "no arguments"

print "the current loglevel is ", "$loglevel$"
print "hello"
```

Allerdings muss die *$loglevel$* Variable vor dem Aufruf der ExecPython Sektionn gesetzt werden:

```
DefVar $LogLevel$
set $loglevel$ = getLoglevel
```

Damit wir am Ende in der Lage sind, die Ergebnisse der Ausgabe weiter zu verarbeiten, wird eine String-List Variable erstellt, die über die execPython Sektion folgendermaßen aufgerufen werden kann:

```
DefStringList pythonresult
Set pythonResult = GetOutStreamFromSection('execpython_hello -a "opt a"')
```

8.13.2 Beispiele

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_compare_to_python$ = "on"*

8.14 ExecWith Sektionen

ExecWith Sektionen sind verallgemeinerte *DosBatch* bzw. *ExecPython* Sektionen: Welches Programm den Inhalt der Sektionen ausführt, wird durch einen Parameter beim Sektionsaufruf bestimmt.

Wenn der Aufruf so lautet:

```
‘execPython_hello -a "hello" -b "world"‘
```

so sind

`-a "hello" -b "world"`

Parameter, die vom Phythonskript akzeptiert werden. Mit dem ExecWith Aufruf sieht der gleiche Ausdruck wie folgt aus:

```
‘execWith_hello "python" PASS -a "hello" -b "world" WINST /EscapeStrings‘
```

Die Option `/EscapeStrings` wird in der ExecPython-Sektion automatisch angewendet und bedeutet, dass Backslahes und Konstanten in String-Variablen dupliziert werden, bevor sie das aufgerufene Programm interpretiert.

8.14.1 Aufrufsyntax

Generell haben wir die Aufrufsyntax:

```
ExecWith_SECTION PROGRAM PROGRAMPARAS pass PASSPARAS winst WINSTOPTS
```

Jeder der Ausdrücke *PROGRAM, PROGRAMPARAS, PASSPARAS, WINSTOPTS* können beliebige String-Ausdrücke oder auch einfache String-Konstanten (ohne Anführungszeichen) sein.

Die Schlüsselwörter `PASS` und `WINST` dürfen fehlen, wenn der entsprechende Part nicht existiert.

Es sind zwei *opsi-winst*-Optionen verfügbar:

- `/EscapeStrings`
- `/LetThemGo`

Wie bei ExecPython Sektionen wird die Ausgabe einer ExecWith-Sektion in einer String-Liste über die Funktion `getOutStreamFromSection` erfasst.

Die erste Option legt fest, dass die Backslashes in *opsi-winst*-Variablen und Konstanten dupliziert werden, so dass sie das ausführende Programm in der üblichen Form von Strings in *C*-Syntax vorfindet. Die zweite Option hat den Effekt (wie bei `winBatch` Aufrufen), dass das aufgerufene Programm in einem neuen Thread startet, während der *opsi-winst* mit dem Auslesen des Skripts fortfährt.

Der Inhalt der Sektion wird in eine temporäre Datei (*.bat) gespeichert. Seit Version 4.11.3 wird, wenn als Interpreter *powershell* angegeben ist, die temporäre Datei als `.ps1` gespeichert.

8.14.2 Weitere Beispiele

Der folgende Aufruf verweist auf eine Sektion, die ein *autoit3*-Skript ist, dass auf zu öffnende Fenster wartet (dafür ist die Option `/letThemGo` zu benutzen), um sie dann in der aufgerufenen Reihenfolge zu schließen:

```
ExecWith_close "%SCRIPTPATH%\autoit3.exe" WINST /letThemGo
```

Ein einfacher Aufruf

```
ExecWith_edit_me "notepad.exe"  WINST /letThemGo
```

ruft Notepad auf und öffnet die Sektion als Datei (allerdings ohne die Zeilen die mit einem Semikolon beginnen, da der *opsi-winst* solche Zeilen als Kommentarzeilen interpretiert und vor der weiteren Behandlung der Sektion entfernt).

Für zusätzliche Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_autoit3_test$ = "on"*.

8.15 LDAPsearch Sektion

Eine LDAPsearch Sektion beschreibt eine Suchanfrage an ein LDAP Verzeichnis, die ausgeführt wird und auch die Antwort empfängt (und wenn möglich im Cache speichert).

Bevor wir zu den *opsi-winst* Kommandos übergehen, gibt es erst noch einige Erklärungen zum Syntax von LDAP selbst

8.15.1 LDAP – Protokoll, Service, Verzeichnis

LDAP bedeutet "Lightweight Directory Access Protocol" und ist, wie der Name besagt, ein festgelegter Weg der Kommunikation mit einem Datenverzeichnis.

Dieses Verzeichnis ist für gewöhnlich hierarchisch organisiert. Es ist eine hierarchische Datenbank oder ein Datenbaum.

Ein **LDAP Service** implementiert das Protokoll zum Lesen und Schreiben auf diesem Verzeichnis. Ein Verzeichnis, dass über einen LDAP Service angesteuert werden kann, nennt sich **LDAP directory**.

Für ein Beispiel werfen einen Blick auf einen Bereich eines LDAP Verzeichnisbaums mit Daten aus dem opsi LDAP-Backend (angezeigt im Open Source LDAP-Browser JXPlorer).

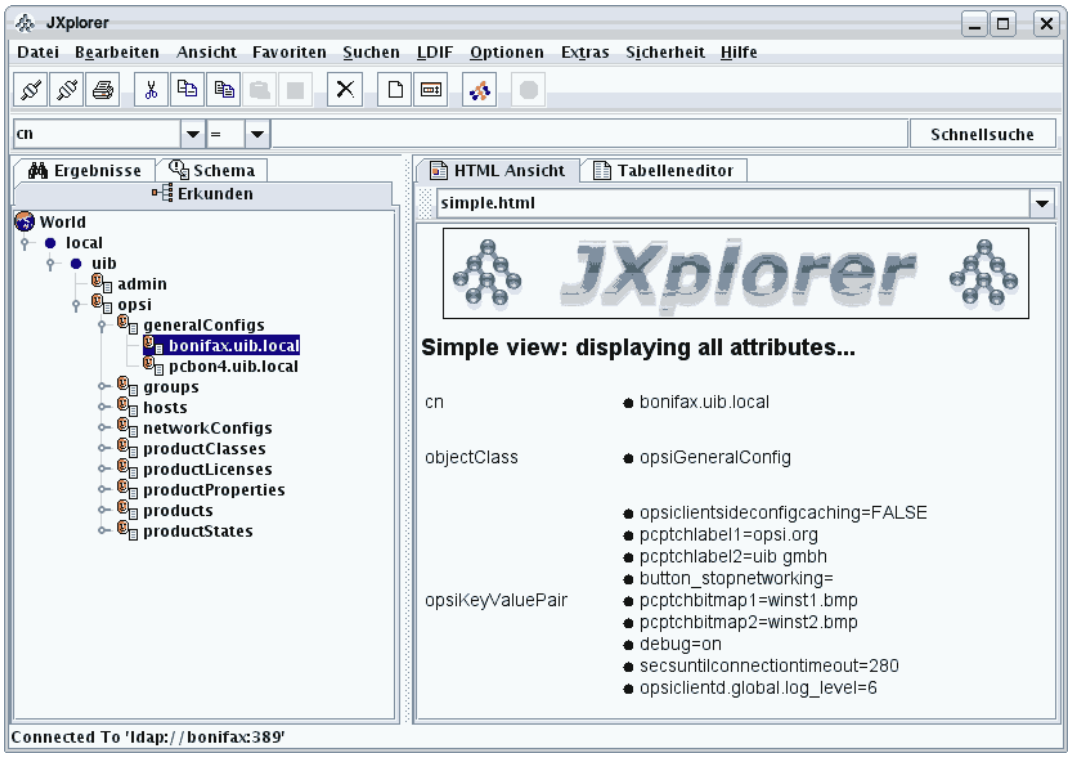

Abbildung 8.4: Ansicht von verschiedenen Bereichen des opsi LDAP Baums

Ein **LDAP search request** ist ein Suchabfrage an das LDAP Verzeichnis über einen LDAP Service. Als Antwort werden verschiedene Inhalte des Datenverzeichnisses zurückgegeben.

Grundsätzlich beschreiben Suchabfragen den Pfad im Verzeichnisbaum, der zu der gewünschten Information führt. Der Pfad ist der **distinguished name** (dn) zusammen gesetzt aus den Namen der Knoten ("relative distinguished names") welche den Pfad bilden. Zum Beispiel:

local/uib/opsi/generalConfigs/bonifax.uib.local

Da jeder Knoten als eine Instanz einer strukturellen Objektklasse konzipiert ist, wird die Pfadbeschreibung in folgender Form ausgegeben: mit Klassentyp (und beginnend mit dem letzten Pfadelement):

cn=bonifax.uib.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local

Der Pfad in einer Abfrage muss nicht notwendigerweise „komplett“ sein und auch nicht zu einem einzelnen Blatt (Teil) des Baumes führen. Im Gegenteil, unvollständige Pfade sind üblich.

Aber auch wenn der Pfad zu einem einzelnen Blatt führt, kann dieses wiederum mehrere Werte enthalten. Jeder Knoten des Baumes hat eine oder mehrere Klassen als Attributtypen. Zu jeder Klasse können ein oder mehrere Werte zugehörig sein.

Bei einem gegebenen Abfragepfad könnten wir uns interessieren für

1. für die Knoten – auch LDAP Objekte genannt – zu welchen der Pfad führt,
2. für die Attribute, die zu den Knoten gehören,
3. und die Werte, die sowohl zu den Objekten wie zu den Attributen gehören.

Offensichtlich ist der Umgang mit der Fülle der Informationen möglicher Antworten die vorrangige Herausforderung bei der Abwicklung von LDAP Abfragen.

Der folgende Abschnitt zeigt eine LDAP Abfrage über den Bereich des LDAP Baums, welcher in der obenstehenden Grafik abgebildet ist.

Beispiel einer LDAP Antwort

Eine *opsi-winst* Sektion `ldapsearch_generalConfigs` ist wie folgt definiert:

```
[ldapsearch_generalConfigs]
targethost: bonifax
dn: cn=generalConfigs,cn=opsi,dc=uib,dc=local
```

Der Sektionsaufruf gibt eine LDAP Antwort zurück, die folgendermaßen aussieht:

```
Result: 0
  Object: cn=generalConfigs,cn=opsi,dc=uib,dc=local
  Attribute: cn
        generalConfigs
  Attribute: objectClass
        organizationalRole
Result: 1
  Object: cn=pcbon4.uib.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
  Attribute: cn
        pcbon4.uib.local
  Attribute: objectClass
        opsiGeneralConfig
  Attribute: opsiKeyValuePair
        test2=test
        test=a b c d
Result: 2
  Object: cn=bonifax.uib.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
  Attribute: objectClass
        opsiGeneralConfig
  Attribute: cn
        bonifax.uib.local
  Attribute: opsiKeyValuePair
        opsiclientsideconfigcaching=FALSE
        pcptchlabel1=opsi.org
        pcptchlabel2=uib gmbh
        button_stopnetworking=
        pcptchbitmap1=winst1.bmp
        pcptchbitmap2=winst2.bmp
        debug=on
        secsuntilconnectiontimeout=280
        opsiclientd.global.log_level=
```

Es gibt nun verschiedene *opsi-winst* Optionen, um die Komplexität der Auswertung der Ergebnisse solcher Anfragen zu reduzieren und zu handhaben.

8.15.2 LDAPsearch Aufrufparameter

Für den Aufruf von LDAPSearch Sektionen sind zwei Typen von Optionen definiert.

- cache options
- output options

Die *cache options* sind:

- `/cache`
- `/cached`
- `/free`
- (no cache option)

Wenn keine cache Option spezifiziert wurde, wird die Antwort der LDAP Suche nicht für zukünftige Anwendung gespeichert.

Bei der `/cache` Option wird die Antwort für zukünftige Auswertungen gespeichert, die `/cached` Option verweist auf die letzte gespeicherte Antwort, welche wiederverwendet wird, statt eine neue Suche zu starten, die `/free` Option löscht die gecachten Antworten (dies ist vor allem bei Suchanfragen mit sehr langen Antworten sinnvoll).

Die output options sind:

- `/objects`
- `/attributes`
- `/values`
- (no output option)

Die Ausgabeoptionen bestimmen die String-Listen, die produziert werden, wenn eine LDAPsearch Sektion über getReturnlistFromSection aufgerufen wird:

- Wenn die Ausgabeoptionen nicht näher spezifiziert werden, wird die komplette LDAP Antwort aufgelistet.
- Die Optionen objects, attributes und values beschränken die Ausgabe entsprechend auf Zeilen zu Objekten, Attributen bzw. Werten in der LDAP Antwort.

Zu beachten ist, dass die ausgegebenen Listen von Attributen nur dann dem richtigen Objekt zu geordnet werden können, wenn die gesamte Ausgabe nur noch ein Objekt enthält. Ebenso sind Werte nur dann dem korrekten Attribut zuordnenbar, wenn nur noch ein Attribut in der Ausgabeliste vorkommt.

Daher wird so vorgegangen, dass eine ursprüngliche Suche immer weiter eingeengt wird bis nur noch ein Objekt bzw. Attribut zurückgegeben wird. Dies kann über entsprechende Count Aufrufe überprüft werden.

Die Einengung der ursprünglichen Suche geht sehr schnell, wenn diese auf der gecachten Antwort durchgeführt wird.

8.15.3 Einengung der Suche

Ein Beispiel soll zeigen, wie die Suche soweit eingeschränkt werden kann, damit ein bestimmtes Ergebnis bei einer Suche im LDAP Verzeichnis erreicht werden kann.

Wir starten mit dem Aufruf von *ldapsearch_generalConfigs* (wie oben beschrieben), fügen den cache Parameter hinzu,

```
ldapsearch_generalconfigs /cache
```

die Abfrage wird ausgeführt und die Antwort für zukünftige Nutzung gespeichert.

Dann gibt der Aufruf

```
getReturnlistFromSection("ldapsearch_generalconfigs /cached /objects")
```

folgende Liste aus

```
cn=generalconfigs,cn=opsi,dc=uib,dc=local
cn=pcbon4.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
cn=bonifax.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
```

Wenn wir die Auswahl im Baumverzeichnis mit

```
[ldapsearch_generalConfigs]
targethost: bonifax
dn: cn=bonifax.ubi.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
```

einschränken und nochmal starten, enthält die Objektliste nur noch folgende Einträge

```
cn=bonifax.uib.local,cn=generalconfigs,cn=opsi,dc=uib,dc=local
```

Die dazugehörige Attributliste enthält drei Elemente:

```
objectclass
cn
opsikeyvaluepair
```

Um die zugehörigen Werte zu einem einzelnen Attribut zu bekommen, muss die Abfrage noch erweitert werden:

```
[ldapsearch_generalConfigs]
targethost: bonifax
dn: cn=bonifax.ubi.local,cn=generalConfigs,cn=opsi,dc=uib,dc=local
attribute: opsiKeyValuePair
```

Das Ergebnis ist eine Attributliste, die nur ein Element enthält. Die Liste mit den zugehörigen Werten sieht wie folgt aus

```
opsiclientsideconfigcaching=false
pcptchlabel1=opsi.org
pcptchlabel2=uib gmbh
button_stopnetworking=
pcptchbitmap1=winst1.bmp
pcptchbitmap2=winst2.bmp
debug=on
secsuntilconnectiontimeout=280
opsiclientd.global.log_level=6
```

Es gibt keine LDAP Mittel um diese Ergebnis noch weiter einzugrenzen!

(Aber die *opsi-winst* Funktion getValue (key, list) (vgl. Kapitel "(Wieder-) Gewinnen von Einzelstrings aus String-Listen") hilft in diesem Fall: z.B. getValue ("secsuntilconnectiontimeout", list) würde die gewünschte Zahl ausgeben).

Mit der Funktion `count` (list) kann überprüft werden, ob die Eingrenzung der Suchabfrage erfolgreich war. In den meisten Fällen ist gewünscht, dass das Ergebnis "1" ist.

8.15.4 LDAPsearch Sektion Syntax

Eine LDAPsearch Sektion beinhaltet die Spezifikationen:

- `targethost:`
  Der Server, der das LDAP Verzeichnis verwaltet/gespeichert wird (service), muss benannt werden.
- `targetport:`
  Wenn der Port eines LDAP Service nicht der voreingestellte Port von 389, muss er an dieser Stelle angegeben werden. Wenn die Spezifizierung nicht erfolgt, wird der Default-Port verwendet.
- `dn:`
  Hier kann der charakteristische Name (distinguished name), der „Suchpfad“, für die Suchanfrage gegeben werden.
- `typesonly:`
  Per Voreinstellung ist der Wert "false", was bedeutet das auch die Werte ermittelt werden.
- `filter:`
  Der Filter für eine LDAP Suche hat eine spezielle LDAP Syntax, die nicht vom *opsi-winst* überprüft wird. Voreingestellt ist "(objectclass=*)".
- `attributes:`
  Durch Kommas werden die Attributnamen in einer Liste getrennt. Die Default-Einstellung ist eine Liste, in der alle Attribute aufgeführt werden.

8.15.5 Beispiele

Ein kurzes und sehr realistisches Beispiel soll am Ende dieses Abschnittes aufgeführt werden:

$founditems$ ist eine StringList Variable und $opsiClient$ ist eine String-Variable. Der Aufruf von *getReturnlistFromSection* liefert die Ergebnisse. Das nachfolgende Codefragment gibt das eindeutige Ergebnis für $opsiDescription$ zurück, wenn dieses existiert. Es vermeldet einen Fehler, wenn die Suche ein unerwartetes Ergebnis zurück gibt:

```
set $opsiClient$ = "test.uib.local"
set $founditems$ = getReturnlistFromSection("ldapsearch_hosts /values")

DefVar $opsiDescription$
set $opsiDescription$ = ""
if count(founditems) = "1"
  set $opsiDescription$ = takeString(0, founditems)
else
  if count(founditems) = "0"
    comment "No result found")
  else
    logError "No unique result for LdAPsearch for client " + $opsiclient$
  endif
endif


[ldapsearch_hosts]
targethost: opsiserver
targetport:
dn: cn=$opsiclient$,cn=hosts,cn=opsi,dc=uib,dc=local
typesOnly: false
filter: (objectclass=*)
attributes: opsiDescription
```

Für weitere Beispiele beachten Sie das Produkt *opsi-winst-test* und dort den Bereich *$Flag_winst_ldap_search$ = "on"*.

Kapitel 9

64 Bit-Unterstützung

Der *opsi-winst* ist ein 32 Bit-Programm. Damit sich auch 32 Bit-Programme auf 64 Bit-Systemen normal arbeiten können, gibt es für 32 Bit-Programme sowohl in der Registry als auch im Dateisystem Spezialbereiche auf die Zugriffe umgeleitet werden, die sonst in 64 Bit vorbehaltenen Bereichen landen würden.

So wird ein Zugriff auf *c:\windows\system32* umgelenkt auf *c:\windows\syswow64*.

Aber ein Zugriff auf *c:\program files* wird **nicht** umgelenkt auf *c:\program files (x86)*

So wird ein Registry Zugriff auf *[HKLM\software\opsi.org]* umgelenkt auf *[HKLM\software\wow6432node\opsi.org]*.

opsi-winst installiert daher als 32 Bit-Programm Skripte, die unter 32 Bit laufen, auch in 64 Bit-Systemen korrekt.

Für die Installation von 64 Bit-Programmen liefern einige alte Konstanten wie '%ProgramFilesDir%'´ für 64 Bit-Programme die falschen Werte. Daher gibt es ab winst Version 4.10.8 folgende Neuerungen:

In der Regel kann (und sollte) nun explizit angegeben werden, wohin geschrieben und woher gelesen werden soll. Dazu gibt es drei Varianten:

32
: Explizit 32 Bit

64
: Explizit 64 Bit. Wenn es das nicht gibt, dann architekturspezifisch.

SysNative
: Entsprechend der Architektur auf dem das SKript läuft.

Entsprechend gibt es zusätzlichen Konstanten:

Tabelle 9.1: Konstanten

Konstante	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%`
: sollte in Zukunft besser gemieden werden.

%ProgramFiles32Dir%
sollten Sie verwenden, wenn Sie explizit 32 Bit-Software installieren wollen.

%ProgramFiles64Dir%
sollten Sie verwenden, wenn Sie explizit 64 Bit-Software installieren wollen.

%ProgramFilesSysnativeDir%
sollten Sie verwenden, wenn Sie auf den Default der Architektur zugreifen wollen.

Für den Zugriff auf eigentlich 64 Bit-Software vorbehaltene Bereiche kennt der *opsi-winst* folgende zusätzlichen Befehle:

- `GetRegistrystringvalue32`
- `GetRegistrystringvalue64`
- `GetRegistrystringvalueSysNative`
- `FileExists32`
- `FileExists64`
- `FileExistsSysNative`

`Registry`-Sektionen schreiben in den 32 Bit-Bereich der Registry. Ebenfalls werden in `Files`-Sektionen Zugriffe auf *c:\windows\system32* umgelenkt.

Für `Registry` und `Files` Sektionen gibt es daher nun die Aufrufparameter:

- `/32Bit`
  Das ist der Default. Schreibzugriffe werden in die 32 Bit-Registry bzw. das 32 Bit-Systemverzeichnis gelenkt.
- `/64Bit`
  Schreibzugriffe werden in die 64 Bit-Registry bzw. das 64 Bit-Systemverzeichnis gelenkt. Gibt es diesen nicht, so wird der architekturspezifische Zweig verwendet.
- `/SysNative`
  Schreibzugriffe werden in den architekturspezifischen Zweig der Registry bzw. des Systemverzeichnisses gelenkt.

Für `DosBatch`, `DosInAnIcon` usw. gilt das selbe, nur das die Parameter durch das Schlüsselwort `winst` abgergernzt werden müssen.

Beispiel:

```
DosInAnIcon_do_64bit_stuff winst /64Bit
```

Als weitere Möglichkeit für explizite 64 Bit-Operationen wird bei der Installation des opsi-client-agent die Datei *c:\windows\system32\cmd.exe* nach *c:\windows\cmd64.exe* kopiert. Durch den Aufruf von SKripten mit dieser *cmd64.exe* im Rahmen von `ExecWith` Sektionen können beliebige 64 Bit-Operationen ausgeführt werden.

Beispiele:
File handling:

```
if $INST_SystemType$ = "64 Bit System"
        comment ""
        comment "-------------------------------------"
        comment "Testing: "
        message "64 Bit redirection"
        Files_copy_test_to_system32
        if FileExists("%System%\dummy.txt")
                comment "passed"
        else
```

```
                LogWarning "failed"
                set $TestResult$ = "not o.k."
        endif
        ExecWith_remove_test_from_system32 'cmd.exe' /C
        Files_copy_test_to_system32 /64Bit
        if FileExists64("%System%\dummy.txt")
                comment "passed"
        else
                LogWarning "failed"
                set $TestResult$ = "not o.k."
        endif
        ExecWith_remove_test_from_system32 '%SystemRoot%\cmd64.exe' /C
endif
```

Registry Handling:

```
message "Write to 64 Bit Registry"
if ($INST_SystemType$ = "64 Bit System")
        set $ConstTest$ = ""
        set $regWriteValue$ = "64"
        set $CompValue$ = $regWriteValue$
        Registry_opsi_org_test /64Bit
        ExecWith_opsi_org_test "%systemroot%\cmd64.exe" /c
        set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByWinst")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
        set $ConstTest$ = GetRegistryStringValue64("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByReg")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
        set $regWriteValue$ = "32"
        set $CompValue$ = $regWriteValue$
        Registry_opsi_org_test
        ExecWith_opsi_org_test "cmd.exe" /c
        set $ConstTest$ = GetRegistryStringValue("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByWinst")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
                comment "failed"
        endif
        set $ConstTest$ = GetRegistryStringValue("[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\test]
    bitByReg")
        if ($ConstTest$ = $CompValue$)
                comment "passed"
        else
                set $TestResult$ = "not o.k."
```

```
                comment "failed"
        endif
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
```

Kapitel 10

Kochbuch

In diesem Kapitel sind Skript-Beispiele zusammengestellt, wie durch den Einsatz verschiedener *opsi-winst* Funktionen gewisse Aufgaben, die sich in ähnlicher Weise immer wieder stellen, bewältigt werden können.

10.1 Löschen einer Datei in allen Userverzeichnissen

Seit *opsi-winst* Version 4.2 gibt es für diese Aufgabe eine einfache Lösung: Wenn etwa die Datei alt.txt aus allen Userverzeichnissen gelöscht werden soll, so kann der folgende Files-Sektions-Aufruf verwendet werden:

```
files_delete_Alt /allNtUserProfiles

[files_delete_Alt]
delete "%UserProfileDir%\alt.txt"
```

Für ältere *opsi-winst* Versionen sei hier noch ein Workaround dokumentiert, der hilfreiche Techniken enthält, die eventuell für andere Zwecke dienen können. Folgende Zutaten werden benötigt:

- Eine DosInAnIcon-Sektion, in der ein dir-Befehl die Liste aller Verzeichnisnamen produziert.
- Eine Files-Sektion, die das Löschen der Datei alt.txt in einem bestimmten Verzeichnis anstößt.
- Eine String-Listen-Verarbeitung, die alles miteinander verknüpft.

Das Ganze kann z.B. so aussehen:

```
[Actions]

; Variable für den Dateinamen:
DefVar $loeschDatei$
set $loeschDatei$ = "alt.txt"

; Variablendeklaration für die String-Listen
DefStringList list0
DefStringList list1

; Einfangen der vom Dos-dir-Befehl produzierten Zeilen
Set list0 = getOutStreamFromSection ('dosbatch_profiledir')

; Aufruf einer Files-Sektion für jede Zeile
for $x$ in list0 do files_delete_x
```

```
; Und hier die beiden benötigten Spezialsektionen:
[dosbatch_profiledir]
dir "%ProfileDir%" /b

[files_delete_x]
delete "%ProfileDir%\$x$\$LoeschDatei$"
```

10.2 Überprüfen, ob ein spezieller Service läuft

Wenn wir überprüfen wollen, ob eine spezieller Service (beispielsweise der "opsiclientd") läuft und ihn, falls er nicht läuft, starten wollen, müssen wir folgendes Skript verwenden.

Um eine Liste der laufenden Services angezeigt zu bekommen, müssen wir das Kommando
`net start`
in einer DosBatch Sektion starten und das Ergebnis in der $list0$ erfassen. Wir gleichen die Liste ab und iterieren die Elemente, um zu sehen, ob der spezielle Service beinhaltet ist. Wenn er nicht da ist, wird er gestartet.

```
[Actions]
DefStringList $list0$
DefStringList $list1$
DefStringList $result$
Set $list0$=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
```

10.3 Skript für Installationen im Kontext eines lokalen Administrators

In manchen Situationen kann es sinnvoll oder notwendig sein, ein *opsi-winst* Skript als lokal eingeloggter Benutzer auszuführen anstatt wie üblich im Kontext eines Systemdienstes. Beispielsweise kann es sein, dass Softwareinstallationen, die vom *opsi-winst* aus aufgerufen werden, zwingend einen Benutzerkontext benötigen oder dass bestimmte Dienste, die für den Installationsvorgang wichtig sind, erst nach dem Login zur Verfügung stehen.

MSI-Installationen, die eine lokalen User benötigen lassen, sich häufig durch die Option *ALLUSERS=1* dazu "überreden" auch ohne aus zukommen. Beispiel:

```
[Actions]
DefVar $LOG_LOCATION$
Set $LOG_LOCATION$ = "c:\tmp\myproduct.log"
winbatch_install_myproduct

[winbatch_install_myproduct]
msiexec /qb ALLUSERS=2 /l* $LOG_LOCATION$ /i %SCRIPTPATH%\files\myproduct.msi
```

Eine andere aufwendigere Möglichkeit dieses Problem zu lösen, ist einen administrativen User temporär anzulegen und diesen zur Installation des Programms zu verwenden.

Dazu gehen Sie wie folgt vor:

Erzeugen Sie ein neues Produkt auf Basis des Produkte opsi-template-with-admin Legen im Verzeichnis *install\productid* ein Verzeichnis *localsetup* an. Verschieben Sie die gesamten Installationsdateien Ihres Produktes in das Unterverzeichnis *localsetup* des erzeugten Produktes.

Sorgen Sie dafür, dass Ihr setup-script einen **Reboot** auslöst. Dazu fügen Sie am besten direkt unter `[Actions]` die Zeile `ExitWindows /Reboot` ein:

```
[Actions]
ExitWindows /Reboot
```

Das nachfolgende abgedruckte *opsi-winst* Skript-Template erzeugt temporär den gewünschten Benutzerkontext, führt in ihm eine Installation aus und beseitigt ihn schließlich wieder. Für die Verwendung sind die folgende Variablen zu setzen:

- der richtige Wert für die Variable `$Productname$`,
- der richtige Wert für die Variable `$ProductSize$` und
- der richtige Wert für die Variable `$LocalSetupScript$` (Name des eigentlichen setup-scriptes)

Das Skript führt im Einzelnen folgende Schritte aus:

- Anlegen eines lokalen Administrator opsiSetupAdmin;
- Sichern des bisherigen Autologon-Zustands;
- Eintragen des opsiSetupAdmin als Autologon-User;

- Installationsdateien auf den Client kopieren (wohin steht in `$localFilesPath$`), dort befindet sich das Installationsskript, das als lokaler Benutzer ausgeführt werden soll;
- RunOnce-Eintrag in der Registry anlegen, der den *opsi-winst* mit dem lokalen Skript als Argument aufruft;
- Neustart des Client (damit die Änderungen an der Registry Wirkung haben);
- *opsi-winst* startet und führt ExitWindows /ImmediateLogout aus: durch den Autologon meldet sich nun automatisch der Benutzer opsiSetupAdmin an, es wird der RunOnce-Befehl ausgeführt;
- nun läuft die Installation ganz normal, jedoch am Ende des Skripts muss zwingend neugestartet werden (also mit ExitWindows /ImmediateReboot), da sonst die Oberfläche des momentan eingeloggten Users opsiSetupAdmin mit Administratorrechten(!) freigegeben wird;
- nach dem Reboot wird wieder aufgeräumt (alten Zustand von Autologon wiederherstellen, lokale Setup-Dateien löschen, Benutzerprofil von opsiSetupAdmin löschen).

Wie man sieht, gliedert sich die Installation in 2 Bereiche: ein Skript, das als Service ausgeführt wird, alles zum lokalen Login vorbereitet und später wieder aufräumt (Masterscript) und ein Skript, dass als lokaler Administrator ausgeführt wird und die eigentliche Setup-Routine für das Produkt enthält (Localscript).

Achtung
Erfordert das Localscript mehr als nur einen Reboot, muss auch das Masterscript verändert bzw. um diese Anzahl von Reboots erweitert werden. Solange das Localscript nicht fertig ist, muss das Masterscript ein ExitWindows /ImmediateLogout ausführen, um die Kontrolle an das Localscript zu übergeben. Der RunOnce-Eintrag muss dann immer wieder neu gesetzt werden. Ebenso müssen Username und Passwort des Autologins nach jedem Reboot neu gesetzt werden.

Es gibt (ab opsi 4.0.2-2) einen direkten Zugang vom lokalen Skript auf die Produkteigenschaften und das Lizenzmanagement mit den gewohnten Funktionen.

Es kann Produktinstallationen (also aus dem Localscript heraus) geben, die Schlüssel in der Registry verändern, die vorher vom Masterscript gesichert und am Ende durch dieses wieder überschreiben werden. In diesem Fall muss die Wiederherstellung der Werte im Masterscript unterbunden werden.

Das Localscript läuft unter eingeloggtem Administrator Account. Wenn hier nicht Keyboard/Maus gesperrt werden, besteht für den Anwender die Möglichkeit das Skript zu unterbrechen und Administrator zu werden.

Es gibt daher im Template ein Property *debug*. Steht dieses auf dem Default (=false) so werden Tastatur und Maus gesperrt und keine Passwörter geloggt. Während der Skriptentwicklung und den Tests kann dieses Property auf *true* gesetzt werden.

Das Passwort des temporären opsiSetupAdmin wird in nachfolgenden Beispiel durch die Funktion RandomStr bestimmt.

Wichtig
Verwenden Sie aktualisierte Versionen des folgenden Beispiels aus dem Templateprodukt: opsi-template-with-admin.

```
; Copyright (c) uib gmbh (www.uib.de)
; This sourcecode is owned by uib
; and published under the Terms of the General Public License.

; TEMPLATE for
; Skript fuer Installationen im Kontext eines temporaeren lokalen Administrators
; installations as temporary local admin
; see winst_manual.pdf / winst_handbuch.pdf
```

```
;
; !!! Das lokale Installations-Skript, das durch den temporaeren lokalen Admin ausgefuehrt wird
; !!! (sein Name steht in $LocalSetupScript$), muss mit dem Befehl
; !!! exitWindows /Reboot
; !!! enden
;

; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
; Vorarbeiten/Voraussetzungen/Doku pruefen wie in Winsthandbuch
;  Skript fuer Installationen im Kontext eines lokalen Administrators
; !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


[Actions]
requiredWinstVersion >= 4.11.3.3
setLogLevel=7
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.ins"
; ----------------------------------------------------------------

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"
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"
```

10.4 XML-Datei patchen: Setzen des Vorlagenpfades für OpenOffice.org 2.0

Das Setzen des Vorlagenpfades kann mit Hilfe der folgenden Skriptteile erfolgen:

```
[Actions]
; ....

DefVar $oooTemplateDirectory$
;-----------------------------------------------------------------------
;set path here:

Set $oooTemplateDirectory$ = "file://server/share/verzeichnis"
;-----------------------------------------------------------------------
;...

DefVar $sofficePath$
Set $sofficePath$= GetRegistryStringValue ("[HKEY_LOCAL_MACHINE\SOFTWARE\OpenOffice.org\
    OpenOffice.org\2.0] Path")
DefVar $oooDirectory$
Set $oooDirectory$= SubstringBefore ($sofficePath$, "\program\soffice.exe")
DefVar $oooShareDirectory$
Set $oooShareDirectory$ = $oooDirectory$ + "\share"

XMLPatch_paths_xcu $oooShareDirectory$+"\registry\data\org\openoffice\Office\Paths.xcu"
; ...


[XMLPatch_paths_xcu]
OpenNodeSet
- error_when_no_node_existing false
- warning_when_no_node_existing true
- error_when_nodecount_greater_1 false
- warning_when_nodecount_greater_1 true
- create_when_node_not_existing true
- attributes_strict false

documentroot
all_childelements_with:
elementname: "node"
attribute:"oor:name" value="Paths"
all_childelements_with:
elementname: "node"
```

```
attribute: "oor:name" value="Template"
all_childelements_with:
elementname: "node"
attribute: "oor:name" value="InternalPaths"
all_childelements_with:
elementname: "node"

end

SetAttribute "oor:name" value="$oooTemplateDirectory$"
```

10.5 XML-Datei einlesen mit dem opsi-winst

Wie bereits im vorangehenden Kapitel "XML-Datei patchen" beschrieben, lassen sich auch XML-Dateien mit dem *opsi-winst* einlesen. Hier soll nun exemplarisch gezeigt werden, wie man die Werte eines bestimmten Knotens ausliest. Als Quelle dient dazu folgende XML-Datei:

```
<?xml version="1.0" encoding="utf-16" ?>
<Collector xmlns="http://schemas.microsoft.com/appx/2004/04/Collector" xmlns:xs="http://www.w3.
    org/2001/XMLSchema-instance" xs:schemaLocation="Collector.xsd" UtcDate="04/06/2006 12:28:17"
    LogId="{693B0A32-76A2-4FA0-979C-611DEE852C2C}"  Version="4.1.3790.1641" >
   <Options>
      <Department></Department>
      <IniPath></IniPath>
      <CustomValues>
      </CustomValues>
   </Options>
   <SystemList>
      <ChassisInfo Vendor="Chassis Manufacture" AssetTag="System Enclosure 0" SerialNumber="EVAL"
    />
      <DirectxInfo Major="9" Minor="0"/>
   </SystemList>
   <SoftwareList>
      <Application Name="Windows XP-Hotfix - KB873333" ComponentType="Hotfix" EvidenceId="256"
    RootDirPath="C:\WINDOWS\$NtUninstallKB873333$\spuninst" OsComponent="true" Vendor="Microsoft
    Corporation" Crc32="0x4235b909">
         <Evidence>
            <AddRemoveProgram DisplayName="Windows XP-Hotfix - KB873333" CompanyName="Microsoft
    Corporation" Path="C:\WINDOWS\$NtUninstallKB873333$\spuninst" RegistryPath="
    HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KB873333"
    UninstallString="C:\WINDOWS\$NtUninstallKB873333$\spuninst\spuninst.exe" OsComponent="true"
    UniqueId="256"/>
         </Evidence>
      </Application>
      <Application Name="Windows XP-Hotfix - KB873339" ComponentType="Hotfix" EvidenceId="257"
    RootDirPath="C:\WINDOWS\$NtUninstallKB873339$\spuninst" OsComponent="true" Vendor="Microsoft
    Corporation" Crc32="0x9c550c9c">
         <Evidence>
            <AddRemoveProgram DisplayName="Windows XP-Hotfix - KB873339" CompanyName="Microsoft
    Corporation" Path="C:\WINDOWS\$NtUninstallKB873339$\spuninst" RegistryPath="
    HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\KB873339"
    UninstallString="C:\WINDOWS\$NtUninstallKB873339$\spuninst\spuninst.exe" OsComponent="true"
    UniqueId="257"/>
         </Evidence>
      </Application>
```

```
   </SoftwareList>
</Collector>
```

Möchte man nur die Elemente und deren Werte aller „Application“-Knoten auslesen, kann man dies machen mit folgendem Code (nur Ausschnitt):

```
[Actions]
DefStringList $list$

...

set $list$ = getReturnListFromSection ('XMLPatch_findProducts '+$TEMP$+'\test.xml')
for $line$ in $list$ do Sub_doSomething

[XMLPatch_findProducts]
openNodeSet
        ; Knoten „Collector" ist der documentroot
        documentroot
        all_childelements_with:
          elementname:"SoftwareList"
        all_childelements_with:
          elementname:"Application"
end
return elements

[Sub_doSomething]
set $escLine$ = EscapeString:$line$
; hier kann man nun diese Elemente in $escLine$ bearbeiten
```

Hier sieht man auch eine weitere Besonderheit. Es sollte vor dem Benutzen der eingelesenen Zeilen erst ein EscapeString der Zeile erzeugt werden, damit enthaltene Sonderzeichen nicht vom *opsi-winst* interpretiert werden. Die Zeile wird nun gekapselt behandelt, sonst könnten reservierte Zeichen wie $,%,“ oder \’ leicht zu unvorhersehbaren Fehlfunktionen führen.

,

10.6 Einfügen einer Namensraumdefinition in eine XML-Datei

Die *opsi-winst* XMLPatch-Sektion braucht eine voll ausgewiesenen XML Namensraum (wie es im XML RFC gefordert wird). Aber es gibt XML Konfigurationsdateien, in denen „nahe liegende“ Elemente nicht deklariert werden (und auslesende Programme, die auch davon ausgehen, dass die Konfigurationsdatei entsprechend aussieht).

Besonders das Patchen der meisten XML/XCU Konfigurationsdateien von OpenOffice.org erweist sich als sehr schwierig. Um dieses Problem zu lösen hat A. Pohl (Vielen Dank!) die Funktionen XMLaddNamespace und XMLremoveNamespace entwickelt. Die Funktionsweise ist im folgenden Beispiel demonstriert:

```
DefVar $XMLFile$
DefVar $XMLElement$
DefVar $XMLNameSpace$
set $XMLFile$ = "D:\Entwicklung\OPSI\winst\Common.xcu3"
set $XMLElement$ = 'oor:component-data'
set $XMLNameSpace$ = 'xmlns:xml="http://www.w3.org/XML/1998/namespace"'

if XMLAddNamespace($XMLFile$,$XMLElement$, $XMLNameSpace$)
  set $NSMustRemove$="1"
endif
;
```

```
; now the XML Patch should work
; (commented out since not integrated in this example)
;
; XMLPatch_Common $XMLFile$
;
; when finished we rebuild the original format
if $NSMustRemove$="1"
  if not (XMLRemoveNamespace($XMLFile$,$XMLElement$,$XMLNameSpace$))
    LogError "XML-Datei konnte nicht korrekt wiederhergestellt werden"
    isFatalError
  endif
endif
```

Es ist zu beachten, dass die XML Datei so formatiert wird, dass der Element-Tag-Bereich keine Zeilenumbrüche enthält.

Kapitel 11

Spezielle Fehlermeldungen

- Keine Verbindung mit dem opsi-Service
  Der *opsi-winst* meldet "... cannot connect to service".

Hinweise auf mögliche Probleme gibt die dazu angezeigte Nachricht:

Socket-Fehler #10061, Verbindung abgelehnt
: Möglicherweise läuft der Service nicht

Socket-Fehler #10065, Keine Route zum Host
: Keine Netzwerkverbindung zum Server

HTTP/1.1. 401 Unauthorized
: Der Service antwortet, akzeptiert aber das Passwort nicht.