# PC Software Installation mit wInst

## - der opsi Windows Installer -

### Anleitung für Entwickler von Installationsskripten


Revisionsstand: 25.05.09

uib gmbh (www.uib.de)

# Inhaltsverzeichnis

## Versionen und Änderungen diese Handbuchs (Revisonshistorie)

### **wInst** version 4.8.6 (opsi version 3.4)

Neue boolesche Funktion opsiLicenseManagementEnabled (vgl. Abschnitt 6.7.3)

Neue String Funktionen DemandLicenseKey, FreeLicenseKey (Abschnitt 6.3.12),

getLastServiceErrorClass, getLastServiceErrorMessage (Abschnitt 6.3.13)

### **wInst** version 4.8.4 (opsi version 3.3.1)

Neue Version der Funktion check option -V für Kopieraktionen; bedeutet das die Überprüfung der Version nur im Hinblick auf die Dateien im Target Verzeichnis erfolgt (vgl. Abschnitt 7.1.2)

### **wInst** version 4.8.1 (opsi version 3.3.1)

Neue Konstante %installingProduct% (Abschnitt 5.2.3 (iv)).

Für das Lizenzmanagement: Neue String Funktionen demandLicenseKey, freeLicenseKey

### **wInst** version 4.7.4 (opsi version 3.3.1)

Neue Betriebssystemsversion Funktionen GetMSVersionInfo (die Versionsinformation wird als WinApi bereitgestellt)

GetSystemType (für XP und Vista; mögliche Werte '64 Bit System' oder ' x86 System'

### **wInst** version 4.6.0 (opsi version 3.3)

**wInst** hat eine neue Oberfläche, die editierbar ist (skinnable **wInst**, Abschnitt 3.2)

### **wInst** version 4.5.9 (opsi version 3.2 updated)

Neue Stringlisten Funktionen getLocaleInfoMap und getFileVersionMap (Abschnitt 6.4.1)

Neue Strinfunktion getValue($key, $map) für eine String $key und eine Stringliste $map (Abschnitt 6.4.4)

Neuer modifizierter copy-Befehl -c (vgl. Abschnitt 7.1.3)

Neue Konstante %ipAddress%, %ipName% (vgl. Abschnitt 5.2.3).

Neue Stringfunktion getLastExitCode. Diese Funktion gibt den ExitCode – oder den ErrorLevel – des letzten winbatch-Aufrufs zurück. Neue String Funktionen abschneiden.

Neue Kommando für die erste Sektion: sleepSeconds, markTime, diffTime (vgl. Abschnitt 6.6)

Neuer Sektionstyp ExecWith (vgl. Abschnitt 7.15)

### **wInst** version 4.5.6 (opsi version 3.2 updated)

Neue Variante des ExitWindows Kommandos (/ShutdownWanted, vgl. Abschnitt 6.9).

### **wInst** version 4.5 (opsi version 3.2)

Neuer Sektionstyp execPython (Abschnitt 7.14) wurde eingeführt. Wenn python auf dem System installiert ist, wird die python.exe aufgerufen und die Sektion als ein Python-Skript ausgelesen. Um das Python-Skript mit dem winst-Skript zu verweben, gibt es neue Konstanten wie %opsiserviceURL%, %opsiserviceUser%, %opsiservicePassword%, %hostID%, %logfile% (vgl. 5.2.3) und eine neue String-Funktion getLoglevel (kurz loglevel; vgl. 6.3.11).

### **wInst** version 4.4 (benötigt für opsi version 3.1)

Neuer Sektionstyp opsiServiceCall (Abschnitt 7.13) für eine direkte oder interaktive Verbindung, bei dem das Passwort mitgeliefert wird. Außerdem besteht die Möglichkeit mit dem opsi Service zu kommunizieren. Neue Funktionen sind XMLaddNamenspace und XMLremoveNamespace (siehe dazu Abschnitt 6.7.3 und im Kochbuch 8.6).

### **wInst** version 4.3 (benötigt für opsi version 3.0)

Anhang (Abschnitt 9.1) Start-Fehlermeldung, wenn die Verbindung zum opsi-Service fehlschlägt.

Korrigierte Beschreibung der *WaitForProcessEnding* Option für die Winbatch-Sektion.

Die Konfiguration der PCs kann über den opsi Service (opsi Version 3.0) ermittelt werden (Abschnitt 2 dieses Handbuchs).

Mit der neuen Funktion *requiredWinstVersion* (vgl. Abschnitt 6.3.3) kann überprüft werden welche **wInst** Version benötigt wird.

### **wInst** Version 4.2 (ausgeliefert mit opsi Version 2.5)

Unterstützt die Zustandsbeschreibung "failed" (Abschnitt 6.10)

Neue RandomStr-Funktion (vgl. Abschnitt 6.2.9, 8.3)

Pseudofunktion von EscapeString (Abschnitt 6.3.2)

Für Files-Sektionen mit der Option /allNtUserProfiles kann die neue Variable %UserProfileDir% benutzt werden (Abschnitt 7.1.2)

**wInst**–Konstanten können jetzt in der *sub*-Sektion verwendet werden (Abschnitt 6.1)

Optional gibt es eine neue LogLevel-Syntax (Abschnitt 6.1.2)

### **wInst** Version 4.1

Neuer Parameter /WaitForProcessEnding für *WinBatch*-Aufrufe (Abschnitt 7.9)
Parameter /ImmediateLogout für ExitWindows-Kommandos wurde eingeführt (Abschnitt 6.9, 8.3)

Syntaxvariante /regedit für Registry-Sektionen (Abschnitt 7.12)

Neue Stringlistenfunktion loadUnicodeTextFile (Abschnitt 6.4.1, 7.12.4)

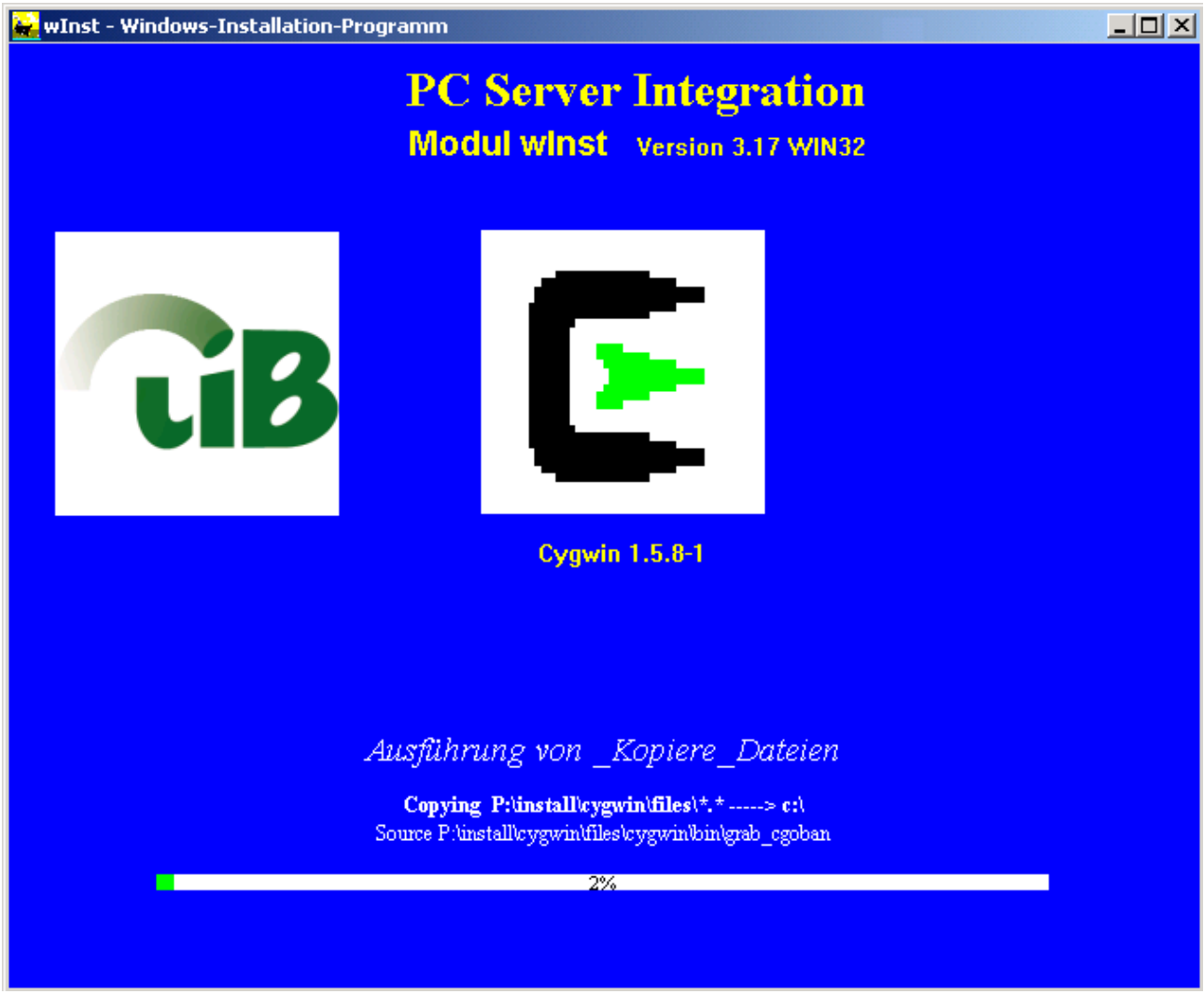
Aufruf eines Unterprogrammes mit einem Stringlistenausdruck als Parameter (Abschnitt 6.8.1)

## **wInst** Version 4.0

Stringlistenverarbeitung (Abschnitt 5.4, 6.4, 8.2 ,...)

Erfassen von Ausgaben der DosBatch/Shell Aufrufen als Stringliste (Abschnitt 6.4.4)

Patchen von XML-Dateien (Abschnitt 7.7)




# 1 Windows Installer

Das Open-Source-Programm **`wInst`** (oder **Windows Installer**) fungiert im Kontext des Systems ***opsi*** – Offene 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.

**`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 **wInst** Skript basiert, bietet verschiedene Vorteile im Vergleich zu Verfahren, die auf Commandshell-Aufrufen beruhen (z. B. Kopieren etc.):

- **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.

***Die wInst-Oberfläche im Test- und Dialogmodus***

# 2 Aufrufparameter

**wInst** kann ja nach Kontext und Verwendungszweck mit unterschiedlichen Parametern gestartet werden.

Es existieren folgende Syntaxvarianten des Aufrufs:

(1) Anzeigen der Varianten:

**wInst /?**

**wInst /h[elp]**

(2 ) Ausführung eines Skripts:

**wInst** ***scriptfile*** **[** ***[/logfile] logfile ]***
***[/batch*** **|** ***/ini winstconfigfilepath]***
***[/parameter parameterstring]***

(3) Auslesen der PC Konfiguration durch denn opsi Service und entsprechende Umsetzung, ab **wInst 4.3**

**wInst** ***/opsiservice [opsiserviceurl]***
***[/clientid clientname]***
***[/username username]***
***[/password password]***
***[[/logfile] logfile]***
***[/parameter parameterstring]]***

(4) Abarbeitung des Software-Profils für einen PC (opsi classic)

**wInst /pcprofil**
**[*PC_configuration_file* [[/logfile] *logfile*]]**
**[/parameter *parameterstring*]**

Generelle Erläuterungen:

- Default Name für die Logdatei ist **C:\tmp\instlog.txt**
- Der ***parameterstring***, angekündigt durch die Option "**/parameter**", wird an das jeweile aufgerufene **wInst** Skript (über die Stringfunktion `ParamStr` ) übergeben.

Erläuterungen zu (2):

- Die Anwendung der Option **/batch** bewirkt, dass nur die Batch-Oberfläche angezeigt wird, die keine Möglichkeiten für Benutzereingaben bietet. Bei Aufruf ohne den Parameter **/batch** erscheint die Dialog-Oberfläche. Mit ihr ist die interaktive Auswahl von Skript- und Logdatei möglich (in erster Linie für Testzwecke gedacht).

- Die Verwendung des Parameters ***/ini*** gefolgt von ***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 ***winstconfigfilepath*** nur ein Verzeichnis benennt (mit "\" abgeschlossen), wird als Dateiname **WINST.INI** verwendet.

Erläuterungen zu (3):

- Ist keine **opsiserviceurl** angegeben, wird als URL verwendet:

  **https://depotserver:4447**

  wobei der **depotserver** aus der depoturl in der Registry ermittelt wird.

- Default für **clientid** ist der Computername.

Erläuterungen zum Format (4):

- In opsi classic werden der PC-Konfigurations-Datei - im Folgenden auch als "Profildatei" angesprochen - die *PC-spezifischen* Informationen über gewünschte Installationen entnommen. Fehlt die Dateiangabe, wird per Default **P:\PCPatch\%PCNAME%.ini** als Konfigurationsdatei verwendet, wobei **%PCNAME%** eine entsprechend vorher zu belegende Environment-Variable ist.

- In der PC-Konfigurationsdatei steht, welche Anwendungen installiert werden sollen. Der Pfad zu dem **wInst**-Skript, das die jeweilige Installation steuert, wird dabei der Datei **pathnams.ini** entnommen, die standardmäßig in p:\pcpatch steht.

- Die Skripte werden automatisch im Batchmodus abgearbeitet.

In der PC-Konfigurationsdatei bzw. durch den opsi service kann eine abweichende Logdatei vorgegeben werden.

Die erforderliche Sektion in der PC-Konfigurationsdatei hat, wenn z.B. die Datei im Verzeichnis n:\tmp unter dem Namen xxx.log abgelegt werden soll, folgende Form

- [winst]

  **Logdateiname=n:\tmp\xxx.log**

# 3 Weitere Konfigurationsoptionen

## 3.1 Zentrales Logging von Fehlermeldungen

**wInst** kann die wichtigsten Logging-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.
- Hat die Variable `remoteerrorlogging` den Wert 1, so sucht das Programm im configshare, Unterverzeichnis pcpatch\pclog, die Datei $pcname$.err und schreibt in sie fortlaufend alle Informationen.
- 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`.

  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)
```

## 3.2 Skinnable **wInst**

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

# 4 Das wInst-Skript

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

In diesem Abschnitt ist der Aufbau eines **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.

## 4.1 Ein Beispiel

In ihren äußeren Form ähneln die **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 **wInst**-Skript (hier mit einer Fallunterscheidung für verschiedene Betriebssystem-Varianten) könnte etwas so aussehen:

```
 [Initial]
Message "Installation von Mozilla"
LogLevel=2

[Aktionen]
;Betriebssystem feststellen
DefVar $0S$
Set $OS$ = GetOS

;Welche NT-Version?
DefVar $NTVersion$
;Wird nur bestimmt, wenn Betriebssystem der NT-Familie angehört

if $OS$ = "Windows_95"
  Sub_install_win95

else
  Set $NTVersion$ = GetNTVersion
  ; hat die Werte "NT4" oder "Win2k"  oder "WinXP"
  ; oder "Win NT " + majorVersion + "." + minorVersion

  if ( $NTVersion$ = "NT4" ) or ( $NTVersion$ = "Win2k" )
     sub_install_winnt
  else
     if ( $NTVersion$ = "WinXP" )
       sub_install_winXP
     else
       stop "Keine unterstützte Betriebssystem-Version"
     endif
  endif
endif


[sub_install_win95]
Files_Kopieren_95
WinBatch_Setup

[sub_install_winNT]
Files_Kopieren_NT
WinBatch_Setup

[sub_install_winXP]
Files_Kopieren_XP
WinBatch_SetupXP
```

```
[Files_Kopieren_95]
copy "%scriptpath%\files_win95\*.*" "c:\temp\installation"

[Files_Kopieren_NT]
copy "%scriptpath%\files_winnt\*.*" "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?

## 4.2 Primäre und sekundäre Unterprogramme des **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 **wInst**-Skripte ist festgelegt, dass die beiden Sektionen mit den Titeln `[Initial]` und `[Aktionen]` (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.

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),
- die **Aktionen**-Sektion (die der Initial-Sektion folgen soll und wie diese nur einmal in einem Skript stehen kann), sowie
- **Sub**-Sektionen (Unterprogramme der **Aktionen**-Sektion, die auch deren Syntax und Funktionslogik erben).

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`**-Sektionen,
- **`DosInAnIcon/ShellInAnIcon`**-Sektionen,
- **`Registry`**-Sektionen
- **`Patches`**-Sektionen,
- **`PatchHosts`**-Sektionen,
- **`PatchTextFile`**-Sektionen,
- **`StartMenu`**-Sektionen,
- **`ProgmanGroups`**-Sektionen (nicht mehr aktuell),
- **`IdapiConfig`**-Sektionen,
- **`XMLPatch`**-Sektionen,

Im Detail wird Bedeutung und Syntax der unterschiedlichen Sektionstypen in den Abschnitten 5 und 6 behandelt.

## 4.3 Stringausdrücke im `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":

  ```
  $NtVersion$
  ```

  kann – sofern der Variable zuvor ein entsprechender Wert zugewiesen wurde - für "`Windows_NT`" 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.

  ```
  GetOs
  ```

  Dies liefert auf einem NT-System wieder den Wert "`Windows_NT`" (anders als bei einer Variablen wird der Wert aber bei jeder Verwendung des Funktionsnamens neu bestimmt).

- Durch einen *additiven Ausdruck*, der einfache Stringwerte 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 6.3).

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.

Nun zunächst genauer zu Definition und Verwendung von Variablen und Konstanten:

# 5 Definition und Verwendung von Variablen und Konstanten im **wInst**-Skript

## 5.1 Allgemeines

Variable und Konstanten erscheinen im Skript als "Wörter", die vom **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 **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 **wInst** das Skript findet und ausführt. Dies könnte beispielsweise `"p:\install\produkt"` sein. Man müsste dann

  **"%ScriptPath%"**

  in das Skript schreiben, wenn man den Wert

  `"p:\install\produkt"`

  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 Stringfunktionen).

  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 Stringlisten**
  werden mit **`DefStringList`** deklariert. Eine Stringlistenvariable kann ihren Inhalt, also eine Liste von Strings, auf unterschiedlichste Weisen erhalten. Mit Stringlistenfunktionen können die Listen in andere Listen überführt oder als Quelle für Einzelstrings verwendet werden.

Im einzelnen:

## 5.2 Globale Textkonstanten

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

### 5.2.1 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 **`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.

### 5.2.2 Beispiel

**`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 Verzeichnis `system`, das im gleichen Verzeichnis wie das Skript liegt, in das Windows-Systemverzeichnis kopiert werden:

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

### 5.2.3 Liste der bereitgestellten Konstanten

Gegenwärtig sind folgende Konstanten definiert:

(i) Pfade und Verzeichnisse des Systems

- **%AppdataDir%**

  Standardmäßig ab Windows 2000:

  **C:\Dokumente und Einstellungen\%USERNAME%\Anwendungsdaten**

- **%AllUsersProfileDir%**

  Standardmäßig ab Windows 2000:

  **C:\Dokumente und Einstellungen\All Users**

- **%CommonStartMenuPath%**

  Standardmäßig

  **C:\Dokumente und Einstellungen\All Users\Startmenü**

- **%ProfileDir%**

  Standardmäßig ab Windows 2000:

  **C:\Dokumente und Einstellungen**

  Hinweis:

  Nur innerhalb von Files-Sektionen, die mit der Option **/AllNtUserProfiles** aufgerufen werden, wird die entsprechende Variable

  **%UserProfileDir%**

  interpretiert. Sie wird dann der Reihe nach belegt mit dem Namen des Profil-Verzeichnisses der verschiedenen auf dem System existierenden Nutzer.

- **%ProgramFilesDir%**

  Standardmäßig

  **C:\Programme**

- **%Systemroot%**

Name des auf dem PC gültigen Heimatverzeichnisses von Windows (ohne schließenden Backslash) – z. B.

```
c:\windows
c:\winnt
```

- **%System%**

  Name des auf dem PC gültigen Windows-Systemverzeichnisses (ohne schließenden Backslash) z. B.

```
c:\windows\system
c:\winnt\system32
```

- **%Systemdrive%**

  Bezeichnung des Laufwerks, auf dem das Betriebssystem installiert ist.

(ii) **wInst**-Pfade und Verzeichnisse

- **%ScriptPath%**

  Pfad des **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 **wInst**-Skript liegt (inklusive Doppelpunkt).

- **%WinstDir%**

  Pfad (ohne schließenden Backslash), in dem der aktive **wInst** liegt.

- **%Logfile%**

  Der Name der Log-Datei, die der **wInst** benutzt.

(iii)Netzwerkinformationen

- **%Host%**

  (Abgekündigt) Wert der Umgebungsvariablen Host, was den Namen des opsi Servers bedeutet – nicht zu verwechseln mit der **%HostID%** , die den Client-Netzwerknamen bezeichnet.

- **%PCName%**

Wert der Umgebungsvariablen PCName, wenn diese nicht existiert, Wert der Variable Computername (sollte der netbios-Namen des PC's sein).

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

- **%IPAddress%**
  Die Netzwerk IP-Adresse.

- **%Username%**

  Der aktuelle Username, wie er im Netz angemeldet ist.

(iv)Daten für den opsi Service

- **%HostID%**

  Der komplette Domainname des opsi Clients wie er von der Kommandozeile oder Ähnlichem geliefert wird.

- **%opsiserviceURL%**

  Die (normalerweise `https://`) URL des opsi Service.

- **%opsiserviceUser%**

  Die Benutzer ID für die es eine Verbindung zum opsi Service gibt.

- **%opsiservicePassword%**

  Das Benutzerpasswort für die Verbindung zum opsi Service. Das Passwort wird beseitigt wenn sich die Standardfunktionen vom **wInst** einloggen.

- **%installingProduct%**

  Der Produktname (productId) für das der Service das laufende Skript aufruft. In dem Fall dass das Skript nicht über den Service läuft bleibt der Stringeintrag leer.

## 5.3 String- (oder Text-) Variable

### 5.3.1 Deklaration

Stringvariable müssen vor ihrer Verwendung deklariert werden. Die Deklarationssyntax lautet

```
DefVar <variable name>
```

Beispielsweise

```
DefVar $NTVersion$
```

Erklärung:

- Die Variablennamen müssen nicht mit "$" beginnen oder enden, diese Konvention erleichtert aber ihre Verwendung.
- Die Deklaration von Variablen *ist nur in den primären Sektionstypen* (**Initial**- oder **Aktionen**-Sektion sowie **sub**-Sektionen) 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.

### 5.3.2 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 6.3).

```
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"
```

### 5.3.3 Verwendung von Variablen in Stringausdrücken

- 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 sie selbst Teil von **`wInst`**-Stringausdrücken 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:

### 5.3.4 Sekundäre und Primäre Sektion im Vergleich

Wenn eine sekundäre Sektion geladen wird, interpretiert der **`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 ersten Sektion einen verknüpften Ausdruck wie folgt formulieren

```
$MailLocation$ = $MailLocation$ + "\backup"
```

Das bedeutet, dass **$MailLocation$** zuerst einen initialen Wert und dann einen neuen Wert annimmt, indem 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).

## 5.4 Variable für Stringlisten

Variable für Stringlisten müssen vor ihrer anderweitigen Verwendung mit dem Befehl **DefStringList** deklariert werden, z. B.

```
DefStringList SMBMounts
```

Stringlisten 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 6.3 zur Stringlistenverarbeitung.

# 6 Syntax und Bedeutung der primären Sektionen eines **wInst**-Skripts

Wie bereits in Abschnitt 4 dargestellt, zeichnen sich die `Aktionen`-Sektion dadurch aus, dass sie den globalen Ablauf der Abarbeitung eines **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 **wInst**-Skripts.

## 6.1 Die primären Sektionen

In einem Skript können drei Arten primärer Sektionen vorkommen:

- eine **Initial**-Sektion zu Beginn des Skripts,
- danach eine **Aktionen**-Sektion sowie
- (beliebig viele) **Sub**-Sektionen.

`Initial`- und `Aktionen`-Sektion sind bis auf die Reihenfolge gleichwertig (`Initial` sollte an erster Stelle stehen). Zur besseren Verständlichkeit sollen in der `Initial`-Sektion statische Einstellungen und -werte bestimmt werden (wie z. B. der Log-Level), während in der `Aktionen`-Sektion die eigentliche Abfolge der vom Skript gesteuerten Programmaktionen beschrieben ist und als Hauptprogramm eines **wInst**-Skripts gesehen werden kann.

`Sub`-Sektionen sind syntaktisch mit der `Initial`- und der `Aktionen`-Sektion vergleichbar, werden aber über die `Aktionen`-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 an externe

Unterprogramme aufgerufen werden (siehe dazu Abschnitt 6.8).

## 6.2 Parametrisierungsanweisungen für den **wInst**

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

### 6.2.1 Beispiel

```
[Initial]
LogLevel=2
ExitOnError=false
ScriptErrorMessages=on
TraceMode=off
```

Dies bedeutet, dass

- der Detaillierungsgrad der Protokollierung auf Level 2 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:

### 6.2.2 Festlegung der Protokollierungstiefe

Es gibt zwei ähnliche Varianten, um den Log-Level zu spezifizieren:

```
LogLevel = <Zahl>
LogLevel = <STRINGAUSDRUCK>
```

**LogLevel** definiert die Tiefe der Protokollierung der Operationen. Im ersten fall kann die Nummer als Integer Wert oder als Stringausdruck (vgl. Kapitel 6.3) angegeben werden. Im zweiten Fall versucht der **wInst** den Stringausdruck als Nummer auszuwerten.

Es sind sechs Detaillierungsgrade wählbar von -2 bis +3.

Der Wert **LogLevel = 0** (Error Level) bedeutet, dass nur ein summarischer Bericht erzeugt wird. Lediglich Fehler und ungewöhnliche Vorkommnisse werden genau aufgezeichnet.

Bei **LogLevel = 1** (Warning Level) werden auch allgemeine Warnungen - das soll hier heißen, Hinweise auf Abläufe, die möglicherweise nicht beabsichtigt waren – protokolliert.

**LogLevel = 2** (Default) heißt, dass die jede Operationen protokolliert werden.

Bei **Level = 3** werden Programm-Debug-Informationen notiert.

**Level = -1** bleibt die Protokollierung auf Fehler beschränkt.

Eine sehr nützlich Einstellung ist der **LogLevel = -2.** *Jede Protokollierung* (abgesehen von **comments**) wird *abgeschaltet*.

### 6.2.3 Benötigte wInst-Version

Die Anweisung

**requiredWinstVersion <RELATIONSSYMBOL> <ZAHLENSTRING>**

z.B.

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

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

Dieses Feature gibt es selbst erst ab **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 **wInst** Version benutzt werden so lange die erforderliche Version ist mindestens 4.3.

### 6.2.4 Reaktion auf Fehler

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

1. fehlerhafte Anweisungen, die der **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 *pop up-Fenster* für eine baldige Korrektur angezeigt, Ausführungsfehler werden in einer *log-Datei* protokolliert und können später analysiert werden.

Das Verhalten des **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.

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.

### 6.2.5 Vordergrund

**StayOnTop = <Wahrheitswert>**

Mittels **StayOnTop = true** (oder = **on**) kann bestimmt werden, dass im *Batchmodus* das **wInst**-Fenster den Vordergrund des Bildschirms in Beschlag nimmt, sofern kein anderer Task den Vordergrundsrang

beansprucht. Im Dialogmodus hat der Wert der Variable keine Bedeutung.

Vorsicht: 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 **`wInst`** im Vordergrund läuft.

## 6.3 String-Werte, String-Ausdrücke und Stringfunktionen

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.

### 6.3.1 Elementare Stringwerte

Ein elementarer Stringwert ist jede Zeichenfolge, die von doppelten – " – oder von einfachen – ' – Anführungszeichen umrahmt ist:

```
"<Zeichenfolge>"
```

oder

```
'<Zeichenfolge>'
```

Zum Beispiel:

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

### 6.3.2 Strings in Strings („geschachtelte“ Stringwerte)

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"'
```

### 6.3.3 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".`**

### 6.3.4 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 5.3.

Die folgenden Abschnitte zeigen die Variationsmöglichkeiten von String Funktionen.

### 6.3.5 Stringfunktionen 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**

  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"**

- **GetSystemType**

  prüft, ob bei dem System 64 Bit vorausgesetzt werden können. In diesem Fall ist der ausgegebene Wert **"64 Bit System"** sonst **"x86 System"**.

### 6.3.6 Stringfunktionen zur Ermittlung von Umgebungs- und Aufrufparametern

- **EnvVar (STRINGAUSDRUCK)**
  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 **wInst** nach dem optionalen Kommandozeilenparameter **/parameter** folgt. Ist der Kommandozeilenparameter nicht verwendet, liefert **ParamStr** den leeren String.

### 6.3.7 Werte aus der Windows-Registry lesen und für sie aufbereiten

- **GetRegistryStringValue (STRINGWERT)**
  versucht den übergebenen Stringwert 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.

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.)

Wenn **KEY** bzw. die Variable **X** nicht existieren, wird eine Warnung in das Logfile geschrieben und der Leerstring als Defaultwert zurückgegeben.

Die Funktion

- **RegString (STRINGWERT)**
  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\\"**

### 6.3.8 Werte aus Ini-Dateien lesen

Es gibt – aus historischen Gründen – drei Funktionen, um Werte aus Ini-Dateien zu lesen. Seit opsi 3.0 werden die Produkteigenschaften vom opsi Konfigurations-Dämon zurückgegeben (der Abruf erfolgt über eine Konfigurationsdatei oder aus einem anderen Backend Datencontainer).

Im Detail:

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.**

Die nächste Funktion unterstützt eine Schreibweise, die sehr eng an die Schreibweise der Ini-Datei selbst angelehnt ist. Dabei kann allerdings nur der Dateiname durch einen Stringausdruck dargestellt werden, die anderen Größen müssen explizit angegeben sein:

**GetIni ( <Stringausdruck> [ <character sequence> ] <character sequence> )**

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","")
```

Zum Beispiel ermittelt

```
GetIni ("%Systemroot%\win.ini" [Interbase] RootDirectory)
```

den Eintrag, der in der Sektion **[Interbase]** der Windows-Inidatei zur Variable **RootDirectory** enthalten ist.

Die dritte Funktion gibt die PC spezifischen Eigenschaften eines Produktes, dass gerade installiert wird, zurück. (**wInst** läuft im pcprofile Modus). Die Syntax lautet

```
IniVar (STRINGAUSDRUCK)

IniVar ("Schalter")
```

ist in der opsi-Standardumgebung bei der Installation von PRODUKT eine Kurzform für

```
GetValueFromIniFile ("p:\pcpatch\%PCNAME%.ini" "PRODUKT-install",
"Schalter", "")
```

Das heißt:

Ab opsi 3.0 liest die Funktion die client-spezifischen Property-Werte für das aktuell installierte Produkt aus (gleichgültig, ob sie in einer Ini-Datei stehen oder woanders).

Auf diese Weise können PC-spezifische Varianten einer Installation konfiguriert werden.

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

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

Innerhalb des Installationsskript werden die ausgewählten Werte wie folgt abgerufen

```
IniVar ("viewer")
IniVar ("policy")
```

### 6.3.9 Informationen aus etc/hosts entnehmen

- **GetHostsName (STRINGWERT)**
  liefert den Hostnamen zu einer gegebenen IP-Adresse entsprechend den Angaben in der Hosts-Datei (die, falls das Betriebssystem laut Environment-Variable OS "Windows_NT" ist, im Verzeichnis "**%systemroot %\system32\drivers\etc**" gesucht wird, andernfalls in "**C:\Windows**").

  Umgekehrt erhält man mit

- **GetHostsAddr (STRINGWERT)**
  die IP-Adresse zu einem gegebenen Host- bzw. Aliasnamen entsprechend der Auflösung in der Hosts-Datei.

### 6.3.10 Stringverarbeitung

- **ExtractFilePath (STRINGWERT)**
  interpretiert den übergebenen String als Datei- bzw. Pfadnamen und gibt den Pfadanteil (den Stringwert bis einschließlich des letzten “\“ zurück).

  Anstelle der obsoleten Funktion

  ```
  StringSplit (STRINGWERT1, STRINGWERT2, INDEX)
  ```

  sollte jetzt der Ausdruck

  ```
  takeString(INDEX, splitString (STRINGWERT1, STRINGWERT2))
  ```

  verwendet werden (siehe den Abschnitt 6.4 Stringlistenverarbeitung).

  Das Ergebnis ist, dass **STRINGWERT1**in Stücke zerlegt wird, die jeweils durch **STRINGWERT2** begrenzt sind, und das Stück mit Index (Zählung bei 0 beginnend) **INDEX** genommen wird.

  Zum Beispiel ergibt

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

den Wert

```
"share"
```

Denn: Mit „\“ zerlegt sich der vorgegebene Stringwert in die Folge:

Index 0 - „“ (Leerstring), weil vor dem ersten „\“ nichts steht

Index 1 - „“, weil zwischen erstem und zweitem „\“ nichts steht

Index 2 - „server“

Index 3 - „share“

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

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

bedeutet das letzte Element der Stringliste `list1`.

- **SubstringBefore (STRINGWERT1, STRINGWERT2)**
  liefert das Anfangsstück von **STRINGWERT1**, wenn **STRINGWERT2** das Endstück ist.

  Z.B. hat

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

  den Wert

  ```
  "C:\programme\staroffice"
  ```

- **Trim(stringValue)**

  Schneidet Leerzeichen am Anfang und Ende des **stringValue** ab.

### 6.3.11 Weitere Stringfunktionen

- **RandomStr**
  liefert Zufallsstrings (der Länge 10), die aus Klein- und Großbuchstaben sowie Ziffern bestehen.

### 6.3.12 (String-) Funktionen für die Lizenzverwaltung

- **DemandLicenseKey (poolId [, productId [,windowsSoftwareId]])**
  Über die Funktion `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 String (bzw. String Ausdrücke).

Wenn die **licensePoolId** nicht explizit gesetzt ist bleibt der erste Parameter ein leerer String "". Das gilt auch für die anderen ID's – 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}")
```

- **FreeLicenseKey (poolId [, productId [,windowsSoftwareId]]]**
  Über die Funktion `freeSoftwareLicenseKey` des opsi Services wird die aktuell belegte Lizenz.

  Diese Syntax ist analog zum Syntax **DemandLicenseKey** zu sehen.

### 6.3.13 Abrufen der Fehlerinformationen von Service Aufrufen

Die String Funktion

**getLastServiceErrorClass**

gibt, wie der Name sagt, den Klassen Namen der Fehlermeldung des letzten Service Aufrufs zurück. Wenn der letzte Serviceaufruf keine Fehlermeldung verursacht hat gibt die Funktion den Wert **"None"** zurück.

In ähnlicher Weise gibt die Funktion

**getLastServiceErrorMessage**

die Nachricht bezgl. der letzten Fehlermeldung **"None"**aus. Seit die Nachrichtenstring sich immer weiter ändern, wird für die Logik des Grundskriptes die Verwendung des Klassennamen empfohlen.

Beispiel:

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

```
endif
```

## 6.4 Stringlistenverarbeitung

Eine Stringliste (oder ein Stringlistenwert) ist ein Sequenz eines Stringwertes. Für diese Werte gibt es die Variable der Stringlisten. Sie sind wie folgt definiert

```
DefStringList <VarName>
```

Ein Stringlistenwert ist einer Stringlistenvariable zugeteilt:

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

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

Für die folgenden Beispiele sei generell eine Stringlisten-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 Stringausdruck stehen können.

Wir beginnen mit einer speziellen und sehr hilfreichen Art von Stringlisten: 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).

### 6.4.1 Info Abbild

- **`getFileInfoMap(FILENAME)`**

  findet die Versionsinformationen, die im **`FILENAME`** verborgen sind und schreibt sie in eine Stringlisten Funktion.

  Zur Zeit existieren folgende Schlüssel,

  ```
  Comments
  CompanyName
  ```

```
FileDescription
FileVersion
InternalName
LegalCopyright
LegalTrademarks
OriginalFilename
PrivateBuild
ProductName
ProductVersion
SpecialBuild
```

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 6.4.4).

- **getLocaleInfoMap**

  fragt die Systeminformationen lokal ab und schreibt sie in eine Stringliste.

  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_country_code**
  **default_oem_code_page**
  **default_ansi_code_page**
  **default_mac_code_page**

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 6.4.4). 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
```

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

- **getLocaleInfo** (ABGEKÜNDIGT)

  Abrufen der lokalen Daten die (vermutlich) am meisten interessieren

  - eine „Zwei-Buchstaben“ Namensangabe der default Systemsprache
  - eine „Drei-Buchstaben“ Namensangabe der default Systemsprache (inklusive der Sprachenuntertypen)
  - den englischen Sprachnamen
  - den englischen Namen des Landes
  - den Sprachcode (hexadezimal Werte als String)

Anwendung: Wenn wir den Aufruf definieren und starten

```
DefStringList $languageInfo$
set  $languageInfo$ = getLocaleInfo
```

haben wir 5 Elemente in der Stringliste. In der Log-Datei bekommen wir folgende Information

```
retrieving strings from getLocaleInfo:
    (string   0)DE
    (string   1)DEU
    (string   2)German
    (string   3)Germany
    (string   4)0407
```

Wir können nun ein Skript für die bedingte Argumente konstruieren (vgl. Abschnitt 6.7) wie z. B.

```
if takeString(0, $languageInfo$ = "DE")
    ; installiere deutsche Version
else
   if takeString(0, $languageInfo$ = "EN")
     ; installiere englische Version
   endif
endif
```

### 6.4.2 Erzeugung von Stringlisten aus vorgegebenen Stringwerten

- **createStringList (Stringwert0, Stringwert1 ,... )**

  erzeugt eine neue Liste aus den aufgeführten einzelnen Stringwerten, z. B. liefert

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

  die ersten vier Buchstaben des Alphabets.

  Die folgenden beiden Funktionen gewinnen eine Stringliste durch die Zerlegung eines gewöhnlichen Strings:

- **splitString (Stringwert1, Stringwert2)**

  erzeugt die Liste der Teilstrings von **Stringwert1**, die jeweils durch **Stringwert2** voneinander getrennt sind. Z. B. bildet

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

  die Liste

  ```
  "", "", "server", "share", "directory"
  ```

- **splitStringOnWhiteSpace (Stringwert)**

  zerlegt **Stringwert** in die durch "leere" Zwischenräume definierten Abschnitte. Das heißt, z. B.

  ```
  set list1 = splitString ("Status   Lokal     Remote         Netzwerk")
  ```

  liefert die Liste

  ```
  "Status", "Lokal", "Remote", "Netzwerk"
  ```

  unabhängig davon, wie viele Leerzeichen oder ggfs. Tabulatorzeichen zwischen den "Wörtern" stehen.

### 6.4.3 Laden der Zeilen einer Textdatei in eine Stringliste

- **loadTextFile (Dateiname)**

  liest die Zeilen der Datei des (als String) angegebenen Namens ein und generiert aus ihnen eine Stringliste.

  Das Gleiche für Unicode-Dateien macht

- **loadUnicodeTextFile (Dateiname)**

  Die Strings in der Stringliste werden dabei (gemäß den Betriebssystem-Defaults) in 8-Bit-Code konvertiert.

### 6.4.4 (Wieder-) Gewinnen von Einzelstrings aus Stringlisten

Die Zerlegung eines Strings in einer Stringliste lässt sich – z. B. nach vorheriger Transformation (s. den Abschnitt „Transformation von Stringlisten“) – rückgängig machen mit der Funktion

- **composeString (StringListe, LinkString)**

  Zum Beispiel wird, wenn **list1** für die Liste `'a', 'b', 'c', 'd', 'e'` steht, erhält die Stringvariable **line** mittels

  ```
  line = composeString (list1, " | ")
  ```

  den Wert "a | b | c | d | e".

Einen Einzelstring aus einer Liste kann man auslesen mit

- **takeString (Index, List1)**

  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'.

Die folgende Funktion versucht eine Stringliste **list1** als eine Liste aus Zeilen des Formulars

```
key=value
```

auszulesen.

So,

- **getValue (key, list1)**

  schaut in ersten Zeile, wo der String **key** nach dem Gleicheitszeichen folgt 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 notwendig für die Nutzung von **getLocaleInfoMap** und **getFileVersionMap** String list Funktionen (vgl. Abschnitt 6.4.1 und 6.4.2).

### 6.4.5 Stringlisten-Erzeugung mit Hilfe von Sektionsaufrufen

- **retrieveSection (Sektionsname)**

  gibt die Zeilen einer aufgerufene Sektion aus.

- **getOutStreamFromSection (Sectionname)**

  „fängt“ – derzeit bei DosBatch, DosInAnIcon bzw. ShellBatch-Befehlen – die Ausgabe der Kommandozeilenprogramme in der Form einer Stringliste ein. Z.B. liefert der Ausdruck

  ```
  [DosInAnIcon_netuse]
  net use
  ```

  wenn die aufgerufene Sektion definiert ist durch

  ```
  getOutStreamFromSection ('DosInAnIcon_netuse')
  ```

  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 Stringliste zur Verfügung stellt. So kann die Anweisung

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

z.B. eine spezifisch selektierte Liste von Knoten der XML-Datei mimetypes.rdf liefern. Näheres zu `XMLPatch_mime` ist der Dokumentation im Kapitel 7.7 zu entnehmen.

Oder die Liste der opsi Clients wird von mit den Hinweisen des folgenden opsi Service Aufrufs erstellt (vgl. Kapitel 7.13)

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

wo

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

### 6.4.6 Transformation von Stringlisten

Eine Teilliste einer vorgegebenen Liste erhält man mit folgender Funktion:

- **getSubList (Startindex, Endindex, list)**

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'`.

- **`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'`.

### 6.4.7 Iteration durch Stringlisten

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

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

- **`for %s% in Liste do Anweisung`**

Dabei wird **`%s%`** durch diesen Ausdruck und nur für diese Stelle als Stringvariable 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.

Vorsicht: 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 Stringvariable 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.

Weitere Beispiele finden sich im Kochbuch-Kapitel im Abschnitt 8.2.

## 6.5 Spezielle Kommandos

- **Killtask <STRINGAUSDRUCK>**

  stoppt alle Prozesse, in denen das durch **STRINGAUSDRUCK** bezeichnete Programm ausgeführt wird.

  Z.B.

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

## 6.6 Anweisungen für Information und Interaktion

- **Message <STRINGAUSDRUCK>**

  bzw.

- **Message = <*Buchstabenfolge*>**

  bewirkt, dass in der Batch-Oberfläche des **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).

  Beispiel:

  ```
  Message "Installation von Mozilla"
  ```

  Dagegen wird in

- **ShowMessageFile <STRINGAUSDRUCK>**

  der Stringausdruck als Dateiname interpretiert und versucht die Datei als Text zu lesen und in einem Textfenster sichtbar zu machen. Z.B. könnte mit

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

  eine "Nachricht des Tages" angezeigt werden.

Mit der Anweisung

- **ShowBitMap [/<ZIFFER>] [<DATEINAME>] [<BILDUNTERSCHRIFT>]**

wird die Bilddatei (BMP- oder PNG-Format, 160x160 Pixel) in der Batch-Oberfläche an der durch **ZIFFER** (1 – 3 möglich) bezeichneten Position angezeigt. Eine **BILDUNTERSCHRIFT** kann ebenfalls hinzugefügt werden.

**<location index>** ist eine `<Sequence von Kennziffern>` - in diesem Fall ist es zu diesem Zeitpunkt nur eine Position wie `1, 2, 3.`

**<DATEINAME>** und **<BILDUNTERSCHRIFT>** sind String Ausdrücke.

Zum Beispiel kann man

```
ShowBitmap /3 "%scriptpath%\" + $ProduktName$ + ".bmp"  "$ProduktName$"
```

aufrufen, um eine spezifisches Produkt-Bild an die Position 3 im Fenster darzustellen.

Wenn der Namensparameter fehlt, wird das Bild an der angegebene Position gelöscht.

Mit

- **comment <STRINGAUSDRUCK>**

bzw.

- **comment = <Zeichensequenz>**

wird einfach der Wert des Stringausdrucks (bzw. Zeichensequenz ) als Kommentar in die Protokolldatei eingefügt.

Zusätzlich Fehlermeldungen bzw. Warnungen in der Protokolldatei werden mit den Anweisungen

- **LogError <STRINGAUSDRUCK>**

oder

- **LogError = <Zeichensequenz>**

bzw.

- **LogWarning <STRINGAUSDRUCK>**

oder

- **LogWarning = <Zeichensequenz>**

  erzeugt.

  Die folgenden Statements dienen debug-Prozessen

- **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.

  Demgegenüber wird bei

- **Stop <STRINGAUSDRUCK>**

  bzw.

- **stop = <Zeichensequenz>**

  der STRINGAUSDRUCK (bzw. Zeichensequenz, die möglicherweise auch leer ist) angezeigt und um Bestätigung gebeten, dass der Programmablauf abbrechen soll.

- **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 auf.

## 6.7 Bedingungsanweisungen (if-Anweisungen)

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

### 6.7.1 Beispiele

Nochmal das Beispiel zum Betriebssystem von vorher:

```
DefVar $OS$
Set $OS$ = GetOS
DefVar $NTVersion$

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

  if ( $NTVersion$ = "NT4" ) or ( $NTVersion$ = "Win2k" )
     sub_install_winnt
  else
     if ( $NTVersion$ = "WinXP" )
       sub_install_winXP
     else
       stop "Keine unterstützte Betriebssystem-Version"
     endif
  endif

endif
```

### 6.7.2 General Syntax

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. Es ist davon abhängig, ob in der Anweisung nach einem **if** Satz (dabei ist es nicht entscheidend, ob in der **if** oder im **else** Teil) eine weitere **if** Anweisung folgen soll.

<*Bedingungen*> sind ***boolesche Ausdrücke***, das heißt logische Ausdrücke, die entweder den Wert **wahr** oder den Wert **falsch** tragen können. Bisher waren diese boolschen Werte in einem **wInst** script nicht deutlich vertreten.

### 6.7.3 Boolesche Ausdrücke

Ein Vergleichsausdruck sieht folgendermaßen aus

**<STRINGAUSDRUCK> <Vergleichszeichen> <STRINGAUSDRUCK>**

An der Stelle **<Vergleichszeichen>** kann eins der folgenden Zeichen stehen

```
<  <=  =  >=  >
```

String Vergleiche im **wInst** sind Fall abhängig.

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

Es gibt einen Vergleichsausdruck für vergleichende Strings *wie (integer) Zahlen*. Wenn irgendeine dieser Werte nicht in eine Zahl übertragen werden kann, wird ein Fehler ausgegeben.

Diese Zahlenvergleichsausdrücke haben die gleich Form wie die Stringvergleichsausdrücke, allerdings wird dem dem Vergleichszeichen ein **INT** vorangestellt:

```
<STRINGAUSDRUCK> INT<Vergleichszeichen> <STRINGAUSDRUCK>
```

So können Ausdrücke wie

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

oder

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

gebildet werden.

Für Beispiele und spezielle Ausdrücke finden sich Funktionen unter Abschnitt 6.3.12.

Boolesche Operator sind **AND**, **OR** und **NOT()** (der Fall ist dabei nicht entscheidend). 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 gleiche Interpretation 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 auch in einen booleschen Ausdruck:

- **FileExists (<STRINGAUSDRUCK>)**

  Die Funktion gibt *wahr* zurück, wenn die genannte Datei oder das Verzeichnis existiert, ansonsten kommt die Antwort *falsch*.

- **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 Stringausdruck). Anderenfalls (oder falls die Datei garnicht existiert) wird *falsch* zurückgegeben.

- **LineBeginning_ExistsIn (stringval, Dateiname)**
  Die Funktion gibt *wahr* zurück, wenn die Zeile **stringval** 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 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 den Abschnitt 8.6.

- **XMLRemoveNamespace(XMLfilename, XMLelementname, XMLnamespace)**

Mit dieser Funktion wird die Definition des XML Namensraum wieder entfernt. Es wird ausgegeben, wenn eine Entfernung erfolgt ist. Dies wird benötigt um zu simulieren, dass die Orginaldatei unverändert geblieben ist (Beispiel im Kochbuch Abschnitt 8.6).

- **HasMinimumSpace (Laufwerksname, Kapazität)**
  gibt *true* zurück, wenn nur noch eine minimal **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 (Fall abhängig).

  Anwendungsbeispiel:

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

  Hilfreich für die Implementierung der Ausgabe der Lizenzschlüssel ist die folgende Funktion

- **opsiLicenseManagementEnabled**

  Es muss ein Umweg gefunden werden, wenn ein Skript von dem Lizenzschlüssel abhängig ist:

```
if opsiLicenseManagementEnabled
   set $mykey$ = DemandLicenseKey ("pool_office2007")
else
   set $mykey$ = IniVar("productkey")
```

## 6.8 Aufrufe von Unterprogrammen

Anweisungen in primären Sektionen, die auf einen Programmtext an anderer Stelle verweisen, sollen hier Unterprogramm- oder Prozeduraufrufe heißen. So "ruft" in obigem Beispiel die Anweisung in der Aktionen-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_SetupXP]**.

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 Stringliste* kann als eine Befehlsliste für einen Sub-Programm Aufruf benutzt werden.

Zur Syntax der Sub-Programm Aufrufe im einzelnen:

### 6.8.1 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* Part ist das Unterprogramm *type specifier*.

  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 6 genauer beschrieben.

- Der *zweite* Part bestimmt, wo und wie die Zeilen des Subprogramms gefunden werden.

  Fall (1): Das Sub-Programm ist eine Zeilenabfolge, die im sich ausführbaren Bereich des **wInst** Scripts 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).

```
sub_install_winXP
```

oder

```
files_copy_winXP
```

Sektionen sind Fall unabhängig wie jeder andere andere String.

Fall (2): Wenn der Programmtyp alleine steht, wird eine Stringliste oder ein Stringausdruck erwartet. Wenn der folgende Ausdruck nicht als Stringlistenausdruck aufgelöst werden kann (vgl. Fall (3)) wird ein String Ausdruck erwartet. Der String wird dann als Dateiname interpretiert. Der **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.

Fall (3): Wenn der Ausdruck auf eine alleinstehenden spezifizierten Sektionstyp folgt kann dieser als ein Stringlistenausdruck aufgelöst werden. Die Stringlistenkomponenten 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 Stringlistenausdruck, 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 Part eines Subsektionsaufrufs:

- Der *dritte* Part eine Aufrufs umfasst spezielle Aufrufsoptionen.

Referenzen für die Aufrufsoptionen beziehungsweise für eine genauere Beschreibung der Sektionsaufrufe finden sich in Kapitel 7.

## 6.9 Reboot-Steueranweisungen

Die Anweisung **ExitWindows** eröffnet die volle Mannigfaltigkeit der Systembefehlen innerhalb eines **wInst** Skripts.

**ExitWindows** stößt einen Reboot des Rechners nach dem Ende der Skript-Abarbeitung an. Im interaktiven Modus wird bei Programmende nachgefragt, ob der Rechner rebootet werden soll. Arbeitet **wInst** im Modus /pcprofil, so wird der Reboot-Wunsch in die Registry eingetragen und vom Programm **pcptch.exe**, als dessen Tochterprozess der **wInst** ausgeführt wird, in der Standardumgebung wieder ausgelesen und ausgeführt. Wenn der Systemaufruf `exitwindows` keinen Erfolg zeigt, wird nach der Kontrolle des opsi Service von der Registry der Aufruf erzwungen. Im Batch-Modus wird der Reboot vom **wInst** selbst angestoßen.

Es gibt verschiedene Varianten einen Reboot in einem **wInst** Skript aufzurufen plus einer abgekündigten Variante. Wir listen sie in ihrer Dringlichkeitsreihenfolge auf:

- **ExitWindows /RebootWanted**
  ABGEKÜNDIGT: vermerkt eine Rebootanfrage eines Skriptes in der Registry, lässt aber das **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 **wInst** die Bearbeitung des laufenden Skriptes beendet hat.

- **ExitWindows /ImmediateReboot**
  unterbricht die normale Ausführung des Skriptes an jeder beliebigen Stelle. Wenn das Kommando `exitwindows` aufgerufen wird, beginnt der **wInst** direkt an dieser Stelle mit dessen Umsetzung. In diesem Kontext garantiert der installierte Preloginloader, dass nach dem Reboot der **wInst** *an der Skriptstelle wiedereinsetzt an der das Skript unterbrochen wurde* (wäre dies nicht der Fall, wurde man in eine Schleife geraten und das Skript nicht fertig

ausgeführt werden).

Die nächste Variante funktioniert ähnlich wie der Ausruf **/ImmediateReboot,** aber beinhaltet einen Logout (Beenden des Skriptes) statt einen Reboot:

- **ExitWindows /ImmediateLogout**
  bewirkt den sofortigen Logout an der Stelle des Skripts, an der der Befehl steht. Dies ist dann sinnvoll, wenn nachfolgend ein automatisches Login (eines anderen Users) folgen soll. Beachten Sie hierzu die Abschnitte 'Installationen mit eingeloggtem User' im Kochbuch (Abschnitt 8.3)

Letztendlich wird ein Herunterfahren des Systems nach der Ausführung aller Skripts gefordert. Zu diesem Zweck dient der **/ShutdownWanted** Parameter:

- **ExitWindows /ShutdownWanted**
  setzt eine Markierung in der Registry, dass der PC nach der Installation aller angefragten Produkte und ihrer Beendigung 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 $OS$
DefVar $Flag$
DefVar $WinstRegKey$
DefVar $RebootRegVar$

set $OS$=EnvVar("OS")

if $OS$="Windows_NT"

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

 if not ($Flag$ = "1")
   ;========================
   ; 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
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
```

## 6.10 Fehlgeschlagene Installation anzeigen

Passieren bei einer Installation Fehler, die zum Fehlschlagen der Installation führen, so wird der Schalter für das Produkt nicht wie sonst in der opsi Umgebung auf `installed` gesetzt, sondern der Produktstatus ist dann `failed`.

Um in einem **`wInst`** Skript eine Installation als gescheitert zu erklären gibt es eine Ausdruck namens

```
isFatalError
```

Die Funktion veranlasst den **`wInst`**, sofort aus der Bearbeitung des Skripts auszusteigen und den Zustand des Produkts für diesen PC auf `failed` zu setzen.

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

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

DefVar $LogErrorMessage$
Set $LogErrorMessage$ = "Nicht genügend Platz auf dem Laufwerk. Erforderlich
sind 200 MB. "
Set $LogErrorMessage$ = $LogErrorMessage$ + $SpaceNeeded"

if not(HasMinimumSpace ("%SYSTEMDRIVE%", $SpaceNeeded"))
  LogError $LogErrorMessage$
  isFatalError
  ; beendet die Skriptausführung und setzt den Prosuktstaus auf failed

 else
  ; die Installation wird gestartet
  ; ...
```

```
endif
```

Es 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 einem Kopierbefehl – 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.

In diesem Zusammenhang gibt es das folgende Skriptbeispiel:

```
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
```

# 7 Sekundäre Sektionen

*Sekundäre Sektionen* können ebenso wie die erste 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.

Der funktionale Kontext wird von der speziellen Syntax des einzelnen Sektionstyps gespiegelt.

Im Detail:

## 7.1 Files-Sektionen

In einer **`Files`**-Sektion stehen Funktionen zur Verfügung, die Kopierbefehlen des Betriebssystems und Verwandtem entsprechen. 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.

### 7.1.1 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 **wInst-**Konstante `%SYSTEMROOT%`).

Die Option `-s` bedeutet dabei, das alle Subdirectories mit erfasst werden, `-v` steht für das Einschalten der Versionskontrolle von Bibliotheksdateien.

### 7.1.2 Aufrufparameter

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%** zur Verfügung, mit der Verzeichnisnamen konstruiert werden können.

Das User-spezische Verzeichnis ist im Fall von **/AllNTUserProfiles** das User-Profilverzeichnis (in der Regel in einem Unterordner von **userappdata** Verzeichnis). 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%\"**

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"
   ```

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

### 7.1.3 Kommandos

Innerhalb einer **Files**-Sektion sind die Anweisungen

- **Copy**
- **Delete**
- **SourcePath**
- **CheckTargetPath**
- **zip**

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.

**zip** dient dem Erzeugen von Archiven.

Im Einzelnen:

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

Die Quelldateien können dabei mittels Joker (”* ” in der Dateimaske) oder auch nur mit einer Pfadangabe bezeichnet werden. `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**
  Gibt es leere Subdirectories im Quellverzeichnis werden sie im Zielverzeichnis ebenfalls leer ("empty") erzeugt
- **v**
  Mit Versionskontrolle:
  Neuere Versionen von Windows-Bibliotheksdateien werden nicht durch ältere Versionen überschrieben (bei unklaren Verhältnissen wird in einem Log-Eintrag gewarnt).
- **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**
  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.
- **w**
  Dateien werden nur überschrieben, wenn sie keinen Schreibschutz haben (das Überschreiben ist "weak", relativ schwach im Vergleich zum Defaultverhalten, dem Ignorieren des Schreibschutzes).
- **n**
  Dateien werden nicht überschrieben.

- **c**
  wenn eine Systemdatei in Benutzung ist, kann sie erst nur nach einem Reboot überschrieben werden. Das **wInst** default-Verhalten ist dabei, dass ein Datei in Benutzung zum Überschreiben beim nächsten Reboot markiert wird UND die **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**
  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>**
- **Delete [-sfd[n]] <Datei(maske)>**

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

  - **s**
    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.
  - **f**
    erzwingt ("force") das Löschen auch von Read-only-Dateien.
  - **d [n]**
    Dateien werden nur gelöscht, sofern sie mindestens n Tage alt sind. Default für n ist 1.

- **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.

- **zip [-s] <Verzeichnis für Archivdateien> <Dateimaske für Quelldateien>**

  **zip** packt alle Dateien gemäß Dateimaske jeweils in eine eigene Archivdatei (im vorgegebenen Verzeichnis). Mit der Option `-s` geschieht dies rekursiv auch für Unterverzeichnisse.

## 7.2 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 **wInst**-Skript erstellt werden, muss man vorsichtig mit den Bezeichnungen umgehen, damit kein Durcheinander entsteht).

### 7.2.1 Beispiele

So lange es noch keine Einträge in der Registry gibt, spielt die Datei **win.ini** eine zentrale Rolle. Sie kann über einen **Patches**-Aufruf editiert werden: In der primären Sektion wird dann eingetragen

```
Patches_WIN.INI "%SYSTEMROOT%\WIN.INI"
```

und die aufgerufene Sektion wird bspw. für den Acrobat Writer näher beschrieben:

```
[Patches_WIN.INI]
set [Devices] Acrobat Distiller=winspool,Ne00:
set [Devices] Acrobat PDFWriter=winspool,LPT1:
set [PrinterPorts] Acrobat Distiller=winspool,Ne00:,15,45
set [PrinterPorts] Acrobat PDFWriter=winspool,LPT1:,15,45
set [Windows] Device=Acrobat PDFWriter,winspool,LPT1:
```

### 7.2.2 Aufrufparameter

Wie im Beispiel beschrieben wird der Name der Dateieigenschaften als ein Parameter des Sub-Programmaufrufs gepachtet

### 7.2.3 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 [] (was hier allerdings nicht bedeutet das die Einträge „syntaktisch optional“ erfolgen können).

Syntax und Funktion der Anweisungen im Einzelnen:

- **add [<Sektionsname>] <Variable1> = <Wert1>**

  Fügt einen Eintrag der Form **<Variable1> = <Wert1>** in die Sektion **<Sektionsname>** 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 [<Sektionsname>] <Variable1> = <Wert1>**

  Setzt einen vorhandenen Eintrag **<Variable1> = <WertX>** in der Sektion **<Sektionsname>** um auf **<Variable1> = <Wert1>**. Existieren mehrere Einträge von **<Variable1>**, wird der erste umgesetzt. Falls kein Eintrag mit **<Variable1>** existiert, wird **<Variable1> = <Wert1>** in der Sektion **<Sektionsname>** erzeugt; existiert die Sektion noch nicht, wird sie zuerst erzeugt.

- **addnew [<Sektionsname>] <Variable1> = <Wert1>**

  Der Eintrag **<Variable1> = <Wert1>** wird in der Sektion **<Sektionsname>** 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 [<Sektionsname>] <Variable1> = <Wert1>**

  Ändert einen vorhandenen Eintrag von **<Variable1>** in der Sektion **<Sektionsname>** auf **<Variable1> = <Wert1>**. Falls **<Variable1>** nicht vorhanden ist, wird Nichts geschrieben.

- **del [<Sektionsname>] <Variable1> = <Wert1>**
  bzw.
- **del [<Sektionsname>] <Variable1>**

  In der Sektion **<Sektionsname>** wird gemäß dem ersten Syntaxschema der Eintrag **<Variable1> = <Wert1>** entfernt. Nach dem zweiten Syntaxschema wird der erste Eintrag von **<Variable1>** aus der angesprochenen Sektion gelöscht, unabhängig vom **Wert** des Eintrags.

- **delsec [<Sektionsname>]**

  Die Sektion **<Sektionsname>** der **.INI**-Datei wird mitsamt ihren Einträgen gelöscht.

- **Replace <Variable1>=<Wert1> <Variable2>=<Wert2>**

  In allen Sektionen der **.INI**-Datei wird der Eintrag **<Variable1>=<Wert1>** durch **<Variable2>=<Wert2>** ersetzt. Zur Anwendung dieses Befehls dürfen Leerzeichen weder um die Gleichheitszeichen stehen noch in **<Wert1>** bzw. **<Wert2>** enthalten sein.

## 7.3 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 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 **PatchHosts** gesucht.

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

In einer **PatchHosts**-Sektion existieren die Anweisungen

- **setAddr**
- **setName**
- **setAlias**
- **delAlias**
- **delHost**
- **setComment**

Beispiel:

```
[PatchHosts#MyHostsPatch]
setAddr ServerNo1 111.111.111.111
setAlias ServerNo1 myServer
```

Dieser Patch sorgt dafür, dass `ServerNo1` künftig mit der IP-Adresse 111.111.111.111 geführt wird und als Alias `myServer` hat.

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>**.

## 7.4 IdapiConfig-Sektionen

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

Diese Funktion ist nur unter Windows verfügbar.

Der Name der Datei, die bearbeitet werden soll, wird beim Aufruf als Parameter übergeben:

```
IdapiConfig_resymesa "c:\idapi\idapi.cfg"
```

Die Sektion könnte dann lauten:

```
[IdapiConfig_resymesa]
alias:resabw
driver:dbase
;parametername=parameterwert
TYPE=Standard
PATH=C:\ReSyMeSa\Daten
DEFAULT DRIVER=dbase
setalias
```

Dabei wird durch

- **alias:<Aliasname>**
  ein Aliasname definiert,
- **driver:<Treibername>**
  der Treiber bestimmt.
- **setalias**
  schreibt am Schluss die Daten in das Konfigurationsfile.

Abhängig vom Treiber kann zuvor eine beliebige Zahl von treiberspezifischen Parametern durch Anweisungen der Form

**<Parametername>=<Parameterwert>**

gesetzt werden.

## 7.5 PatchTextFile-Sektionen

Die flexiblen Mittel, die eine **PatchTextFile**-Sektion bietet, dienen dazu, unterschiedliche Konfigurationsdateien zu patchen, die weder das Format einer ini-Datei noch das hosts-Format aufweisen, jedoch über eine Zeilenstruktur verfügen.

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. Abschnitt 6.7.3) zur Verfügung.

### 7.5.1 Beispiele

Zum Beispiel setzt die folgende **PatchTextFile** Sektion in die Mozilla-Datei die URL auf eine zu wählende Homepage-Adresse:

```
[PatchTextFile_NetscapePref]
GoToTop
FindLine_StartingWith 'user_pref("browser.startup.homepage"'
DeleteTheLine
AddLine 'user_pref("browser.startup.homepage", "http://meinSeite.org");'
```

Speziell für Patches der Mozilla- oder Netscape-User-Präferenzen existiert aber auch eine spezielle Anweisung mit der sich das Beispiel reduzieren lässt auf

```
[PatchTextFile_NetscapePref]
```

```
Set_Netscape_User_Pref ("browser.startup.homepage", "http://meineSeite.org")
```

### 7.5.2 Aufrufparameter

Welche Datei mittels einer **PatchTextFile**-Sektion bearbeitet werden soll, steht beim Aufruf der Sektion in einem Dateiparameter, z. B.

```
PatchTextFile_prefsjs $mailhome$ + "prefs.js"
```

### 7.5.3 Kommandos

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

- **Set_Netscape_User_Pref ("<Präferenzvariable>", "<Wert>")**

  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.

- **AddStringListElement_To_Netscape_User_Pref ("<Präferenzvariable>", "<Werteliste>")**

  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.

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.

Sollen alle Zeilen durchsucht werden, kann das mit der

- **GoToTop**

  getan werden, wobei der 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>]**

  Das Setzten des Zeilenzeigers auf die letzte Zeile erfolgt mit

- **GoToBottom**

  Mit dem folgenden Kommando wird die Zeile gelöscht auf der der Zeilenzeiger steht, sofern sich dort eine Zeile befindet (wenn der Zeilenzeiger oben platziert ist wird nichts gelöscht):

- **DeleteTheLine**

Neu einfügen oder anhängen lassen sich Zeilen mit den drei Anweisungen

- **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.

Dateien werden von den folgenden Anweisungen verarbeitet:

- **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.

## 7.6 LinkFolder-Sektionen

### 7.6.1 Windows

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 namens „acrobat“ im Programme-Folder des allgemeinen Startmenüs (für alle user 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 Betriebssystem-Einträge bestimmt wird, an welchem physikalischen Ort des Dateisystems sie real existieren.

Die Subfolder (bzw. Subfolder-Pfad), in dem dann Links angelegt werden, werden dann 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 gesetzten Links gestartet werden. Es wird mit einer mehrzeiligen Parameterliste angewendet. Sie wird gestartet mit

- **set_link**

und endet mit

- **end_link.**

Dazwischen kann der Link folgendes Format haben:

```
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, der Leerstring,
- Bei fehlender Spezifikation eines **icon_file** dient die Programmdatei als Icon-File.
- Der **icon_index** ist per Default 0.

Achtung: Wenn das referenzierte **target** auf einem momentan nicht erreichbaren Share liegt, werden alle Bestandteile des Pfades auf das

Längenschema 8.3 gekürzt.

Workaround:
- Erzeugen einer korrekten Verknüpfung von Hand zu einem Zeitpunkt, in dem das Laufwerk verbunden ist.
- Kopieren der Link-Datei an einen zur Skriptlaufzeit existenten Ort, z. B. c:\Programme
- Als **`target`** gibt man diese Datei an.

Mit

- **`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.

### 7.6.2 Linux

Allgemein kann man sagen das es nur geringe Unterschiede zu der Windows-Version gibt.

Zulässige virtuelle Folder:

```
desktop, startmenu, startup, desktopdirectory, common_startmenu,
common_startup, common_desktopdirectory
```

Zulässige Parameter in set_link

```
name:            // Titel des Links
target:          // Pfad und Name des Programms
parameters:      // Aufrufparameter des Programms
working_dir:     // Arbeitsverzeichnis für das Programm
icon_file:       // Pfad und Name der Icon-Datei
filename         // Name der Desktop Datei (mit Text)
type             // Typ des Links (siehe unten)
categories       // (optional) ; separate Kategorienliste
genericName      // (optional) Beschreibung (name=mozilla->generic=browser)
```

Im Vergleich zu Windows entfällt der Parameter `icon_index`.

Der Parameter `type` ist erforderlich und hat einen der folgenden Werte: `Application, Link, FSDevice, MimeType.`

`Categories` kann leer bleiben oder durch eine, durch Kommas getrennte Liste aus Kategorien der folgenden Liste, spezifiziert werden:

| Category (Kategorie) | Description (Erklärung) |
|---|---|
| Development | An application for development |
| Building | A tool to build applications |
| Debugger | A tool to debug applications |
| IDE jar -cvf scantodb.jar de uk | IDE application |
| GUIDesigner | A GUI designer application |
| Profiling | A profiling tool |
| RevisionControl | Applications like cvs or subversion |
| Translation | A translation tool |
| Office | An office type application |
| Calendar | Calendar application |
| ContactManagement | E.g. an address book |
| Database | Application to manage a database |
| Dictionary | A dictionary |
| Chart | Chart application |
| Email | Email application |
| Finance | Application to manage your finance |
| FlowChart | A flowchart application |
| PDA | Tool to manage your PDA |
| ProjectManagement | Project management application |
| Presentation | Presentation software |
| Spreadsheet | A spreadsheet |
| WordProcessor | A word processor |
| Graphics | Graphical application |
| 2DGraphics | 2D based graphical application |
| VectorGraphics | Vector based graphical application |
| RasterGraphics | Raster based graphical application |
| 3DGraphics | 3D based graphical application |
| Scanning | Tool to scan a file/text |
| OCR | Optical character recognition application |
| Photography | Camera tools, etc. |
| Viewer | Tool to view e.g. a graphic or pdf file |
| Settings | Settings applications |
| DesktopSettings | Configuration tool for the GUI |
| HardwareSettings | A tool to manage hardware components, like sound cards, video cards or printers |

| Category (Kategorie) | Description (Erklärung) |
|---|---|
| PackageManager | A package manager application |
| Network | Network application such as a web browser |
| Dialup | A dial-up program |
| InstantMessaging | An instant messaging client |
| IRCClient | An IRC client |
| FileTransfer | Tools like FTP or P2P programs |
| HamRadio | HAM radio software |
| News | A news reader or a news ticker |
| P2P | A P2P program |
| RemoteAccess | A tool to remotely manage your PC |
| Telephony | Telephony via PC |
| WebBrowser | A web browser |
| WebDevelopment | A tool for web developers |
| AudioVideo | A multimedia (audio/video) application |
| Audio | An audio application |
| Midi | An app related to MIDI |
| Mixer | Just a mixer |
| Sequencer | A sequencer |
| Tuner | A tuner |
| Video | A video application |
| TV | A TV application |
| AudioVideoEditing | Application to edit audio/video files |
| Player | Application to play audio/video files |
| Recorder | Application to record audio/video files |
| DiscBurning | Application to burn a disc |
| Game | A game |
| ActionGame | An action game |
| AdventureGame | Adventure style game |
| ArcadeGame | Arcade style game |
| BoardGame | A board game |
| BlocksGame | Falling blocks game |
| CardGame | A card game |
| KidsGame | A game for kids |
| LogicGame | Logic games like puzzles, etc |
| RolePlaying | A role playing game |

| Category (Kategorie) | Description (Erklärung) |
|---|---|
| Simulation | A simulation game |
| SportsGame | A sports game |
| StrategyGame | A strategy game |
| Education | Educational software |
| Art | Software to teach arts |
| Construction | |
| Music | Musical software |
| Languages | Software to learn foreign languages |
| Science | Scientific software |
| Astronomy | Astronomy software |
| Biology | Biology software |
| Chemistry | Chemistry software |
| Geology | Geology software |
| Math | Math software |
| MedicalSoftware | Medical software |
| Physics | Physics software |
| Teaching | An education program for teachers |
| Amusement | A simple amusement |
| Applet | An applet that will run inside a panel or another such application, likely desktop specific |
| Archiving | A tool to archive/backup data |
| Electronics | Electronics software, e.g. a circuit designer |
| Emulator | Emulator of another platform, such as a DOS emulator |
| Engineering | Engineering software, e.g. CAD programs |
| FileManager | A file manager |
| Shell | A shell (an actual specific shell such as bash or tcsh, not a TerminalEmulator) |
| Screensaver | A screen saver (launching this desktop entry should activate the screen saver) |
| TerminalEmulator | A terminal emulator application |
| TrayIcon | An application that is primarily an icon for the "system tray" or "notification area" (apps that open a normal window and just happen to have a tray icon as well should not list this category) |

| Category (Kategorie) | Description (Erklärung) |
|---|---|
| System | System application, "System Tools" such as say a log viewer or network monitor |
| Filesystem | A file system tool |
| Monitor | Monitor application/applet that monitors some resource or activity |
| Security | A security tool |
| Utility | Small utility application, "Accessories" |
| Accessibility | Accessibility |
| Calculator | A calculator |
| Clock | A clock application/applet |
| TextEditor | A text editor |
| KDE | Application based on KDE libraries |
| GNOME | Application based on GNOME libraries |
| GTK | Application based on GTK+ libraries |
| Qt | Application based on Qt libraries |
| Motif | Application based on Motif libraries |
| Java | Application based on Java GUI libraries, such as AWT or Swing |
| ConsoleOnly | Application that only works inside a terminal (text-based or command line application) |

## 7.7 XMLPatch-Sektionen

Immer häufiger werden Daten aller Art, insbesondere auch Konfigurationsdaten als XML-Dokument erfasst, d.h. in einer Datei abgelegt, deren Aufbau sich nach der XML-Spezifikation richtet (XML = „Extended Markup Language“) (http://www.w3.org/TR/xml/).

Der **wInst** bietet **XMLPatch**-Sektionen an, um XML-Dokumente zu bearbeiten. Beim Aufruf wird einer **XMLPatch**-Sektion der Name einer XML-Datei übergeben. Zum Beispiel

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

Die Aktionen, die **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.

Vor der Darstellung der Aktionen seien zunächst die verwendeten Begriffe verdeutlicht:

### 7.7.1 Struktur eines XML-Dokuments

Ein XML-Dokument beschreibt die Logik eines „Baums“, der sich ausgehend von einer „Wurzel“ – passenderweise **`document root`** genannt – 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

(7) Text-(Knoten-)Inhalte von Elementen.

Im **`wInst`** ist derzeit die Auswahl nach den Gesichtspunkten (1) bis (3) sowie (7) implementiert:

### 7.7.2 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 **`wInst`** Befehl hierfür lautet

- **`OpenNodeSet`**

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

(i) 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, Abschnitt 8.4):

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

(ii) 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.

(iii)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
```

(iv)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“ sein.

### 7.7.3 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
- zum Setzen von Text

Im Einzelnen:

- **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"
```

Der Befehl

- **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.

Mit

- **DeleteAttribute "Attributname"**

wird das betreffende Attribut von jedem Element der aktuellen Knotenmenge entfernt.

Die Anweisung

- **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 Text-Inhalten eines Elements.

Die beiden Anweisungen lauten

- **SetText "Text"**

sowie

- **AddText "Text"**

Z.B. wird, wenn das betreffende Element im der geöffneten Elementmenge liegt, aus

```
SetText "rtf"
```

durch die Anweisung

```
<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, nur wenn kein Text vorhanden ist, und lässt existierenden Text unberührt.

### 7.7.4 Rückgaben an das aufrufende Programm

Eine **XMLPatch**-Sektion kann angewiesen werden, Stringlisten an das rufende Programm zurückzugeben.

Dazu muss sie in einer primären Sektion mit der Stringlisten-Anweisung **getReturnListFromSection** aufgerufen werden. Die Anweisung kann in einem Stringlisten-Ausdruck verwendet werden, z.B. das Ergebnis einer Stringlisten-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 Stringliste 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 nur 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.

## 7.8 ProgmanGroups-Sektionen

Dieser Sektionstyp ist abgekündigt.

## 7.9 WinBatch-Sektionen

In einer **WinBatch**-Sektion kann jedes Windows-Programm als Anweisung verwendet werden. Wie auch aus dem Windows Explorer heraus können Daten-Dateien, die mit einem Programm verknüpft sind, direkt aufgerufen werden.

Nach dem Kopieren von Programm und angepassten Konfigurationsdateien auf die lokale Platte wird mit dem Aufruf folgender **WinBatch**-Sektion das Setup-Programm gestartet:

```
%systemdrive%\temp\setup.exe
```

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

Per Default wartet **wInst** die Selbstbeendigung jedes angestoßenen Prozesses ab. Dieses Verhalten kann mit dem Parameter **/WaitOnClose** auch explizit definiert werden. Soll dagegen die Abarbeitung des Skripts in einem parallelen Prozess weitergeführt werden, d. h. sofort die nächste Zeile der **WinBatch**-Sektion bzw. die nächste Zeile des übergeordneten Programms abgearbeitet werden, ist die Sektion mit dem Parameter **/LetThemGo** aufzurufen.

Es gibt mehrere ausgeklügelter Optionen für die speziellen Umstände.

Die Parametrisierung **/WaitSeconds [AnzahlSekunden]** modifiziert das Verhalten dahingehend, dass **wInst** jeweils erst nach [**AnzahlSekunden**] die Skriptbearbeitung fortsetzt. Die angegebene Zeit stoppt **wInst** auf jeden Fall. In der Defaulteinstellung wird zusätzlich auf das Ende der angestoßenen Prozesse gewartet. Ist letzteres nicht gewünscht, so kann der Parameter mit dem Parameter **/LetThemGo** kombiniert werden.

Speziellere Wartebedingungen können mit

**/WaitForWindowAppearing [Fenstertitel]**

bzw.

**/WaitForWindowVanish [Fenstertitel]**

formuliert werden. Im 1. Fall wartet **wInst** solange, bis ein Prozess, der sich durch ein mit **[Fenstertitel]** benanntes Fenster kenntlich macht, gestartet ist. Im 2. Fall wartet **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.

Auf das Ende Prozesses kann gewartet werden mit

**/WaitForProcessEnding program**

Dabei kann zusätzlich der Timeout gesetzt werden:

**/WaitForProcessEnding program /TimeOutSeconds seconds**

Beispiel:

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

Die Stringfunktion **getLastExitCode** gibt im Erfolgsfall den ExitCode – oder den ErrorLevel – des letzten Prozessaufrufs der vorausgehenden **WinBatch** Sektion aus.

## 7.10 DOSBatch/ShellBatch-Sektionen

### 7.10.1 Windows

**DOSBatch**-Sektionen (auch **ShellBatch** genannt) sollen in erster Linie dazu dienen, vorhandene Kommandozeilenroutinen für bestimmte Zwecke zu nutzen. **wInst** wartet auf die Beendigung des DOS-Batches, 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.

Die Prozess sind herkömmlich erstellt, was zur Konsequenz hat, dass ein Windows Kommandozeilenfenster erscheint, welche Interaktionen des Benutzers erlaubt.

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

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

```
[Aktionen]
```

```
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 "Hello" und "World".

Sollen die Ausgaben, die von Befehlen einer DosBatch-Sektion kommen, aufgefangen werden, so geschieht dies mittels **getOutStreamFromSection ()** aus der Haupt-Sektion des winst-Skripts (siehe Abschnitt 6.4.4).

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

### 7.10.2 Linux

**ShellBatch**-Sektionen funktionieren analog zu den **DOSBatch** Sektionen unter Windows. Die Hauptunterschiede sind dabei:

Die temporäre Datei wird in `/tmp` abgelegt und wird in einem xterm ausgeführt (xterm -e).

Die Ausgabe der Skripte wird zusätzlich in der Logdatei protokolliert.

## 7.11 DOSInAnIcon/ShellInAnIcon-Sektionen

### 7.11.1 Windows

Der Sektionstyp **DOSInAnIcon** oder **ShellInAnIcon** ist identisch mit der betreffenden **DOSBatch** Syntax und ausführenden Methoden. Allerdings wird der Typ anders dargestellt:

Für **DOSInAnIcon**, einen Shellprozess, wird eine minimierte Darstellung gewählt, was zur Konsequenz hat, dass das Kommando-Fenster „als Icon“ ausgeführt wird. Es erscheint also kein Kommando-Fenster und damit ist auch keine Benutzer Interaktion möglich.

Die Ausgaben der Skriptdatei werden nicht angezeigt, aber in der Logdatei

protokolliert.

### 7.11.2 Linux

Der einzige Unterschied in Linux zwischen der **ShellBatch** und der **ShellInAnIcon** Sektion ist, dass kein xterm-Fenster angezeigt wird.

## 7.12 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 der **wInst**-üblichen Detailliertheit protokolliert werden.

### 7.12.1 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
```

### 7.12.2 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.

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:

- 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 5 in diesem Kapitel).

- Eine weitere Variante des **Registry**-Aufrufs dient dazu, die Patch-Anweisungen für die Registry zu verarbeiten, die im *i*nf-Datei-Standard erstellt sind. Zur Kennzeichnung dient der Parameter **/addReg** (in Anlehnung an die entsprechende Abschnittsbezeichnung in einer inf-Datei)(vgl. Abschnitt 6 in diesem Kapitel).

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

### 7.12.3 Kommandos

Die Syntax der Defaultform einer **Registry**-Sektion ist an der Kommandosyntax anderer Patchoperationen des **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 **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. Es gibt auch den Leerstring als Variablenname; dieser entspricht dem "(Standard)"-Eintrag im Registry-Schlüssel.

  Soll eine Registry-Variable erzeugt oder gesetzt werden, die nicht den Defaulttyp "Registry-String" (**REG_SZ**), 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 Stringwerten, die in **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 Stringwert 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 **<listset_user_Rhino.reg 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

  ```
  HKEY_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 `beset_user_Rhino.reg` 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.

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

wobei `RegistryPathPath` z.B. so aussieht.

```
[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 <Dateiname>`**

  liest eine Datei zeilenweise in einen Multistring ein.

- **`SaveValueToFile <varname> <filename>`**

  exportiert die genannten Werte (String oder MultiSZ) als **`filename`** Rheihe (jeder String bildet eine Reihe).

- **`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>**
  (Nicht verfügbar)

- **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.

### 7.12.4 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 alle Schlüsselnamen relativ interpretiert: 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
```

### 7.12.5 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 Inidateien haben.

Ein 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 **wInst**-Registry-Sektionen).

Diese Anweisungen können aber nun nicht als Sektion innerhalb eine **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` unter Windows XP nicht mehr das Regedit4-Format produziert, sondern ein Format, das durch die erste Zeile

```
"Windows Registry Editor Version 5.00"
```

gekennzeichnet ist.

Windows sieht hier zusätzliche Wertetypen vor. Gravierender ist, dass die Exportdatei ursprünglich in Unicode erzeugt wird. Um sie mit den 8-Bit-Mitteln der Standardumgebung des **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 **wInst** durchführen zu lassen. Dazu lässt sich die Stringlistenfunktion `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].

### 7.12.6 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"
```

## 7.13 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 Stringliste zurückgegeben und dann für die Verwendung in Skripten benutzt werden.

### 7.13.1 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 Stringausdruck**
- **/username Stringausdruck**
- **/password Stringausdruck**

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 angegeben und auch die Option „interactive“ nicht ausgewählt wird (weder bei diesem Skript noch zuvor), wird vorausgesetzt das der **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.

### 7.13.2 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":METHODNAME-STRING
"params":[
        JSON PARAMETER ENTRIES
        ]
```

Die **JSON PARAMETER ENTRIES** sind eine Stringliste (eventuell leer) oder komplizierte JSON Elemente (wie sie für die spezielle Methode benötigt werden). Bspw.

```
opsiservicecall_clientIdsList
```

welche die erforderlichen Methodennamen und die (leere) Parameterliste in der folgendem Sektion setzt:

```
[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 Stringlisten Ausdrucks (vgl. das folgende Beispiel) verwendet werden.

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

Die Verwendung von **GetReturnListFromSection** ist dokumentiert in dem Kapitel zur Stringlistenverarbeitung dieses Handbuchs (Abschnitt 6.4.5).

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"
          ]
```

## 7.14 ExecPython Sektionen

Die **ExecPython** Sektionen basieren auf Shell-Sektionen (ähnlich wie **DosInAnIcon**) welche das – im System installierte – Python Übersetzungsprogramm aufruft. Das Programm übernimmt den Sektionsinhalt als Python Skript und die Sektionen werden Parameter für Parameter für das Skript ausgelesen.

Python bietet als ausgewachsene Programmiersprache definitiv mehr Kodierungsoptionen als der **wInst** beinhaltet und ist daher kraftvoller als ein einfaches Kommando-Shell-Programm. Daher kann es empfehlenswert sein, Python für komplizierte Prozesse zu nutzen. Wenn Datenobjekte dem opsi Service ein Python Skript übermitteln sollen ist es das normale Vorgehen so vorzugehen. Seitdem der opsi Service selbst in Python geschrieben wurde und es auch keine Übersetzung der kodierten Daten gibt wird derart vorgegangen.

### 7.14.1 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"
```

während der Abschnitt wie folgt definiert wird:

```
[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 eine 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.

### 7.14.2 Verflechten eines Python Skripts mit einem wInst Skript

Aktuell ist die **execPython** Sektion dem **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 **wInst** log-Datei geschrieben.
- Der **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 StringListe und dann vom laufenden **wInst** Skript weiter verwendet werden.

Ein Beispiel für die ersten beiden Wege der Verflechtung des Python Skripts mit dem **wInst** Skript werden im Anschluss beschrieben. Es wurde erweitert damit einige der Werte von **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 der Aktionen Sektion gesetzt werden:

```
DefVar $LogLevel$
set $loglevel$ = getLoglevel
```

Damit wir am Ende in der Lage sind, die Ergebnisse der Ausgabe weiter zu verarbeiten, wird eine StringList Variable erstellt, die über die **execPython** Sektion folgendermaßen aufgerufen werden kann:

```
DefStringList pythonresult
Set pythonResult = GetOutStreamFromSection('execpython_hello -a "opt a“')
```

## 7.15 ExecWith Sektionen

**ExecWith** Sektionen sind gewöhnlicher als **ExecPython** Sektionen: Welches Programm die Sektionen ausliest wird durch einen Parameter beim Sektionsaufruf bestimmtl.

Z. B. Wenn der Aufruf so lautet

```
execPython_hello -a "hello" -b "world"
```

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 automatisch in der **ExecPython** Sektion benutzt und bedeutet, dass Backslahes und Konstanten in Stringvariablen dupliziert werden, bevor sie das aufgerufene Programm interpretiert.

### 7.15.1 Aufrufsyntax

Generell haben wir die Aufrufsyntax:

```
ExecWith_SECTION PROGRAM PROGRAMPARAS pass PASSPARAS winst WINSTOPTS
```

Jeder der Ausdrücke **PROGRAM, PROGRAMPARAS, PASSPARAS, WINSTOPTS** sind frei wählbare Stringausdrücke oder eine Stringkonstante (ohne Anführungszeichen).

Die Schlüsselwörter **PASS** und **WINST** dürfen fehlen, wenn der entsprechende Part nicht existiert.

Es gibt zwei weitere **wInst** Optionen zu beachten:

- **/EscapeStrings**

- **/LetThemGo**

Wie bei **ExecPython** Sektionen wird die Ausgabe einer **ExecWith** Sektion in einer Stringliste über die Funktion **getOutStreamFromSection** erfasst.

Die erste Option erklärt, dass die Backslahes in **wInst** variablen und Konstanten sind - ähnlich wie in der Programmiersprache 'C'. Die zweite Option hat den Effekt (für **winBatch** Aufrufe), dass das aufgerufene Programm in einem neuen Thread seine Arbeit beginnt, während der **wInst** mit dem Auslesen des Skripts fortfährt.

### 7.15.2 Mehr 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
```

öffnet einen Texteditor und zeigt die Sektionszeilen an (allerdings keine Zeilen die mit einem Komma starten, da der **wInst** solche Zeilen als Kommentarzeile ausliest und sie eliminiert bevor sie ausgeführt werden).

# 8 Kochbuch

In diesem (im Aufbau begriffenen) Kapitel sind Skript-Beispiele zusammengestellt, wie durch den Einsatz verschiedener **wInst**-Funktionen gewisse Aufgaben, die sich in ähnlicher Weise immer wieder stellen, bewältigt werden können.

## 8.1 Löschen einer Datei in allen Userverzeichnissen

Seit **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 **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 Stringlisten-Verarbeitung, die alles miteinander verknüpft.

Das Ganze kann z. B. so aussehen:

```
; in der Aktionen-Sektion:

; Variable für den Dateinamen:
DefVar $loeschDatei$ = "alt.txt"

; Variablendeklaration für die Stringlisten
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$\$LoeschDatei$"
```

## 8.2 Überprüfen, ob ein spezieller Service läuft

Wenn wir überprüfen wollen, ob eine spezieller Service (beispielsweise mit dem "preloginloader") 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.

```
[Aktionen]
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 $myservice$ = "preloginloader"


comment "============================"
comment "search the list"
; for developping loglevel = 3
; loglevel=3
; in normal use we dont want to log the looping
loglevel = -1
for %s% in list1 do sub_find_myservice
loglevel=2
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
```

## 8.3 Skript für Installationen im Kontext eines lokalen Administrators

In manchen Situationen kann es sinnvoll oder notwendig sein, ein **wInst**-Skript als lokal eingeloggter Benutzer auszuführen anstatt wie üblich im Kontext eines Systemdienstes. Beispielsweise kann es sein, dass Softwareinstallationen, die vom **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=2 dazu 'übereden' auch ohne auszukommen. Besispiel:

```
[Aktionen]
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:

- Legen im Verzeichnis `install\produktname` ein Verzeichnis `localsetup` an.
- Verschieben Sie die gesamten Installationsdateien in dieses Verzeichnis.
- Benennen Sie das Installationsscript von `<produktname>.ins` in `local_<produktname>.ins` um.
- Erzeugen Sie im Verzeichnis `install\produktname` eine neue `<produktname>.ins` mit dem nachfolgenden Inhalt und passen Sie in diesem Script in der **Aktionen**-Sektion die Variablen an (siehe unten). Der Rest des Skriptes bleibt unberührt.
- Sorgen Sie dafür, dass das in `local_<produktname>.ins` umbenannte Skript am Ende einen Reboot auslöst.
  Dazu sollte der letzte ausgeführte Befehl der **Aktionen**-Sektion folgende Zeile sein:
  **ExitWindows /Reboot**

    - **Am Anfang des local_<produktname>.ins Skripts fügen Sie einen Aufruf ein um das Passwort des temporären lokalen Administrators im Autologin zu löschen:**

```
[Aktionen]
Registry_del_autologin
;....

[Registry_del_autologin]
openkey [HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon]
set "DefaultUserName"=""
set "DefaultPassword"=""
```

Das nachfolgende abgedruckte **wInst**-Skript-Template erzeugt temporär den gewünschten Benutzerkontext, führt ihn ihm eine Installation aus und beseitigt ihn schließlich wieder. Für die Verwendung sind die folgende Variablen zusetzen:

- der richtige Wert für die Variable `$Productname$`
- der richtige Wert für die Variable `$ProductSize$`
- `$LockKeyboard$` auf „`true`" um das Keyboard zu sperren

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 **wInst** mit dem lokalen Skript als Argument aufruft;
- Neustart des Client (damit die Änderungen an der Registry Wirkung haben);
- **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*).

Zu beachten:

- 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 neugesetzt werden.
- Es gibt keinen direkten Zugang vom lokalen Skript auf die Produkteigenschaften (üblicherweise erfolgt der Zugang über die Stringfunktion **`IniVar`**) . Wenn Werte benötigt werden, müssen diese über das Masterskript geholt werden und werden ggf. temporär in der Registry gespeichert.
- 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 Script zu unterbrechen und Administrator zu werden.
- Das Passwort des temporären `opsiSetupAdmin` wird in nachfolgenden Beispiel durch die Funktion `RandomStr`.
- Damit die Verarbeitung der Passwörter nicht geloggt wird, wird der LogLevel zeitweise auf -2 gesetzt.

(Eine neuere Version des folgenden Beispiels findet sich unter http://www.opsi.org/opsi_wiki/TemplateForInstallationsAsTemporaryLocalAdmin)

```
; Copyright (c) uib gmbh (www.uib.de)
; This sourcecode is owned by uib
; and published under the Terms of the General Public License.

[Initial]
LogLevel=2
ExitOnError=false
ScriptErrorMessages=on
TraceMode=off


[Aktionen]
DefVar $ProductName$
Set $ProductName$ = "softprod"
DefVar $ProductSizeMB$
Set $ProductSizeMB$ = "20"
DefVar $LocalSetupScript$
Set $LocalSetupScript$ = "local_"+$ProductName$+".ins /batch"
DefVar $LockKeyboard$
; set $LockKeyboard$ to "true" to prevent user hacks while admin is logged in
Set $LockKeyboard$="true"
; Set PasswdLogLevel to -2 to prevent passwords to logged (not working yet)
DefVar $PasswdLogLevel$
Set $PasswdLogLevel$="-2"
DefVar $OpsiAdminPass$
DefStringlist $outlist$

; some variables for the sub sections
DefVar $SYSTEMROOT$
DefVar $SYSTEMDRIVE$
DefVar $ScriptPath$
DefVar $ProgramFilesDir$
DefVar $HOST$
DefVar $AppDataDir$
Set $SYSTEMDRIVE$ = "%SYSTEMDRIVE%"
Set $SYSTEMROOT$ = "%SYSTEMROOT%"
set $ScriptPath$="%ScriptPath%"
set $ProgramFilesDir$="%ProgramFilesDir%"
set $Host$="%Host%"
set $AppDataDir$="%AppDataDir%"
; temp is always useful
DefVar $TEMP$
Set $TEMP$= EnvVar("TEMP")
DefVar $Tmp$
set $Tmp$ = EnvVar("TMP")
;Variables for version of the operating system (OS)-Test
DefVar $OS$
DefVar $MinorOS$
set $OS$ = GetOS
set $MinorOS$ = GetNTVersion

DefVar $RebootFlag$
DefVar $WinstRegKey$
DefVar $RebootRegVar$
DefVar $AutoName$
DefVar $AutoPass$
DefVar $AutoDom$
DefVar $AutoLogon$
```

```
DefVar $AutoBackupKey$
DefVar $LocalFilesPath$
DefVar $LocalWinst$

Set $WinstRegKey$ = "HKLM\SOFTWARE\opsi.org\winst"
Set $RebootFlag$ = GetRegistryStringValue("["+$WinstRegKey$+"] "+"RebootFlag")
Set $AutoBackupKey$ = $WinstRegKey$+"\AutoLogonBackup"
Set $LocalFilesPath$ = "C:\opsi_local_inst"
Set $LocalWinst$ = "c:\opsi\utils\winst32.exe"
  if not ( FileExists ($LocalWinst$) )
   Set $LocalWinst$ =          "%ProgramFilesDir
%\opsi.org\preloginloader\utils\winst32.exe"
  endif

if ($OS$ = "Windows_NT")

 if not (($RebootFlag$ = "1") or ($RebootFlag$ = "2"))
 ;========================
 ; Anweisungen vor Reboot


  if not(HasMinimumSpace ("%SYSTEMDRIVE%", ""+$ProductSizeMB$+" MB"))
   LogError "Nicht genügend Platz auf C: . "+$ProductSizeMB$+" MB auf C: für
"+$ProductName$+" erforderlich."
  else

   ; zeige das Produktbild
   ShowBitmap /3 "%scriptpath%\localsetup\"+$ProductName$+".bmp"
"$ProductName$"

   Message "Preparing "+$ProductName$+" install ..."
   sub_Prepare_AutoLogon

   ; es muss reboot werden – sei sicher, dass das Autologon arbeitet

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

  endif ; genuegend platz
 endif ; Rebootflag = not (1 or 2)
 if ($RebootFlag$ = "1")
  ;========================
  ; Anweisungen nach Reboot
  ; Rebootflag weitersetzen
  Set $RebootFlag$ = "2"
  Registry_SaveRebootFlag
  ; die eigentlichen Anweisungen

  Message "Preparing "+$ProductName$+" install ..."
  Registry_enable_keyboard
  ExitWindows /ImmediateLogout
  ; now let the autologon work
  ; it will stop with a reboot
 endif ; Rebootflag = 1
 if ($RebootFlag$ = "2")
  ;========================
```

```
  ; Anweisungen nach dem zweiten Reboot
  Set $RebootFlag$ = "0"
  Registry_SaveRebootFlag
  ; dieses Abschnitt sollte hier verbleiben, auch wenn nichts zu tun ist
  ; eventuell ist etwas „aufzuräumen“
  Message "Cleanup "+$ProductName$+" install ..."
  sub_Restore_AutoLogon
  ; Das ist das Ende der Installation
 endif ; Rebootflag = 2
else
 LogError "Wir brauchen Windows 2000/XP fuer die Installation mit einem
temporaeren lokalen User"
endif

[sub_Prepare_AutoLogon]
; kopieren des Setup Skripts und Dateien
Files_copy_Setup_files_local
; Lesen des aktuellen Autologon Wertes für die Sicherung
set $AutoName$ = GetRegistryStringValue ("[HKLM\SOFTWARE\Microsoft\Windows NT\
CurrentVersion\Winlogon] DefaultUserName")
; wenn der AutoLogonName ist unser SetupAdminUser, ist etwas nicht richtig
gelaufen
; dann lass uns aufräumen
if ($AutoName$="opsiSetupAdmin")
 set $AutoName$=""
 set $AutoPass$=""
 set $AutoDom$=""
 set $AutoLogon$="0"
else
 set $AutoPass$ = GetRegistryStringValue ("[HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon] DefaultPassword")
 set $AutoDom$ = GetRegistryStringValue ("[HKLM\SOFTWARE\Microsoft\Windows NT\
CurrentVersion\Winlogon] DefaultDomainName")
 set $AutoLogon$ = GetRegistryStringValue ("[HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon] AutoAdminLogon")
endif
; Sichern der AutoLogon Werte
Registry_save_autologon
; Bereite den Admin AutoLogon vor
;LogLevel="$PasswdLogLevel$"
LogLevel=-2
set $OpsiAdminPass$= RandomStr
Registry_autologon
; Erstellen unseres SetupAdminUsers
DosInAnIcon_makeadmin
LogLevel=2
; Entfernen von c:\tmp\winst.bat mit Passwort
Files_remove_winst_bat
; speichern des Setup Skriptes als run once
Registry_runOnce
; blockieren von Keyboard und Maus während der Autologin Admin arbeitet
if ($LockKeyboard$="true")
 Registry_disable_keyboard
endif

[sub_Restore_AutoLogon]
; auslesen der AutoLogon Werte aus dem Backup
set $AutoName$ = GetRegistryStringValue("["+$AutoBackupKey$+"]
```

```
DefaultUserName")
set $AutoPass$ = GetRegistryStringValue("["+$AutoBackupKey$+"]
DefaultPassword")
set $AutoDom$= GetRegistryStringValue("["+$AutoBackupKey$+"]
DefaultDomainName")
set $AutoLogon$= GetRegistryStringValue("["+$AutoBackupKey$+"]
AutoAdminLogon")
; wiederherstellen der Werte
;LogLevel="$PasswdLogLevel$"
LogLevel=-2
Registry_restore_autologon
LogLevel=2
; löschen des Setup Admin User
DosInAnIcon_deleteadmin
; aufräumen des Setup Skriptes, Dateien und Profile
Files_delete_Setup_files_local
; Löschen der Profildirectory
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]
delete -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 Administratoren /ADD opsiSetupAdmin
```

```
[Registry_autologon]
openkey [HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon]
set "DefaultUserName"="opsiSetupAdmin"
set "DefaultPassword"="$OpsiAdminPass$"
set "DefaultDomainName"="localhost"
set "AutoAdminLogon"="1"

[Registry_runonce]
openkey [HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce]
set "opsi_autologon_setup"="$LocalWinst$ $LocalFilesPath$\$LocalSetupScript$"

[Registry_disable_keyboard]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kbdclass]
; disable
set "Start"=REG_DWORD:0x4
;enable
;set "Start"=REG_DWORD:0x1
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Mouclass]
; disable
set "Start"=REG_DWORD:0x4
;enable
;set "Start"=REG_DWORD:0x1

[Registry_enable_keyboard]
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Kbdclass]
; disable
;set "Start"=REG_DWORD:0x4
;enable
set "Start"=REG_DWORD:0x1
openkey [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Mouclass]
; disable
;set "Start"=REG_DWORD:0x4
;enable
set "Start"=REG_DWORD:0x1

[Files_remove_winst_bat]
delete -f c:\tmp\_winst.bat
```

## 8.4 XML-Datei patchen: Setzen des Vorlagenpfades für OpenOffice.org 2.0

Das Setzen des Vorlagenpfades kann mit Hilfe der folgenden Skriptteile erfolgen:

```
[Aktionen]
; ....

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="$oooTemplateDirectory$"
```

## 8.5 XML-Datei einlesen mit dem wInst

Wie bereits im vorangehenden Kapitel 7.7 beschrieben, lassen sich auch XML-Dateien mit dem **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\Uni
nstall\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\Uni
nstall\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):

```
[Aktionen]
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 **`wInst`** interpretiert werden. Die Zeile wird nun gekapselt behandelt, sonst könnten reservierte Zeichen wie $,%,“ oder ' leicht zu unvorhersehbaren Fehlfunktionen führen.

## 8.6 Einfügen einer Namensraumdefinition in eine XML-Datei

Die **`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 erweisen 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. Fehlermeldungen

# 9 Keine Verbindung mit dem opsi-Service

Der **`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.