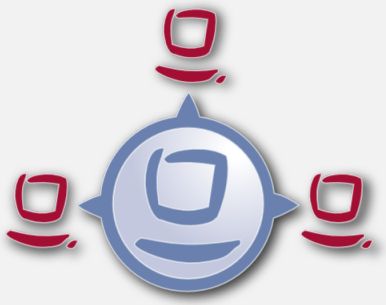


opsi Handbuch opsi-Version 4.0.2

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

Stand: 16.05.2012

Inhaltsverzeichnis

1 Copyright

2 Einführung

2.1 Für wen ist dieses Handbuch?

Diese Handbuch richtet sich an alle, die sich näher für die automatische Softwareverteilung *opsi* interessieren. Der Schwerpunkt der Dokumentation ist die Erläuterung der technischen Hintergründe, um so zu einem Verständnis der Abläufe beizutragen.

Damit soll dieses Handbuch nicht nur den praktisch mit opsi arbeitenden Systemadministrator unterstützen sondern auch im Vorfeld den Interessenten einen konkreten Überblick über opsi geben.

2.2 Konventionen zu Schrift und Grafiken

In *<spitzen Klammern>* werden Namen dargestellt, die im realen Einsatz durch ihre Bedeutung ersetzt werden müssen.

Beispiel: Der Fileshare, auf dem die opsi Softwarepakete liegen, wird <opsi-depot-share> genannt und liegt auf einem realen Server z.B. auf `/opt/pcbin/install`.

Das Softwarepaket: <opsi-depot-share>/ooffice liegt dann tatsächlich unter `/opt/pcbin/install/ooffice`.

Beispiele aus Programmcode oder Konfigurationsdateien stehen in Courier-Schrift und sind farbig hinterlegt.

```
depoturl=smb://smbhost/sharename/path
```

3 Überblick opsi

Werkzeuge zur automatischen Softwareverteilung und Betriebssysteminstallation sind bei größeren PC-Netz-Installationen ein wichtiges Werkzeug zur Standardisierung, Wartbarkeit und Kosteneinsparung. Während die Verwendung solcher Werkzeuge für gewöhnlich mit erheblichen Lizenzkosten einher geht, bietet opsi als Opensource-Werkzeug deutliche Kostenvorteile. Hier fallen nur die Kosten an, die von Ihnen durch tatsächlich angeforderte Dienstleistungen, wie Beratung, Schulung und Wartung, entstehen bzw. soweit kostenpflichtige Module benutzen wollen geringe Kofinanzierungsbeiträge.

Auch wenn Software und Handbücher kostenlos sind, ist es die Einführung eines Softwareverteilungswerkzeuges nie. Um die Vorteile ohne Rückschläge und langwierige Lernkurven nutzen zu können, ist die Schulung und Beratung der Systemadministratoren durch einen erfahrenen Partner dringend geboten. Hier bietet Ihnen uib seine Dienstleistungen rund um opsi an.

Das von uib entwickelte System basiert auf Linux-Servern, über die das Betriebssystem und Software-Pakete auf den PC-Clients installiert und gewartet werden (PC-Server-Integration). Es basiert weitestgehend auf frei verfügbaren Werkzeugen (GNU-tools, SAMBA etc.). Dieses opsi (Open PC-Server-Integration) getaufte System ist durch seine Modularität und Konfigurierbarkeit in großen Teilen eine interessante Lösung für die Probleme der Administration eines großen PC-Parks.

3.1 Erfahrung

opsi ist die Fortschreibung eines Konzepts, das seit Mitte der 90er Jahre bei einer Landesverwaltung auf über 2000 Clients in verschiedenen Lokationen kontinuierlich im Einsatz ist und stetig weiterentwickelt wurde. Als Produkt opsi ist es nun auch einem breiten Kreis von Interessenten zugänglich.

Eine Übersicht registrierter opsi-Installationen finden Sie unter: http://www.opsi.org/map/

3.2 Features von opsi

Die wesentlichen Features von opsi sind:

- automatische Softwareverteilung
- Automatische Betriebssysteminstallation
- Hard- und Softwareinventarisierung
- Komfortable Steuerung über das opsi Managementinterface
- Unterstützung von mehreren Standorten mit Depotservern

3.3 opsi Erweiterungen

- Lizenzmanagement
- MySQL-Backend
- Nagios Connector
- dynamische Depotzuweisung
- Software on Demand
- WAN Erweiterung (Einbindung von Clients hinter langsamen Leitungen)
- User Profile Management: User Profile z.B. in einer Roamig-Profile Umgebung können modifiziert werden.
- OTRS::ITSM Connector (von unserem Partner Cape-IT)

4 opsi-Konfiguration und Werkzeuge

4.1 Übersicht

Die Konfiguration von opsi benötigt eine Datenhaltung. Die externen Komponenten von opsi kommunizieren mit dem opsi-server über einen Webservice. Der Prozess opsiconfd, welcher den Webservice bereitstellt, übergibt die Daten dem Backendmanager, der die Daten in das konfigurierte Backend schreibt.

Dabei unterstützt opsi unterschiedliche Backends:

- File-basiert
- LDAP-basiert
- MySQL-basiert

Beim File-basierten Backend gibt es für jeden Datentyp einen Typ von Ini-Dateien:

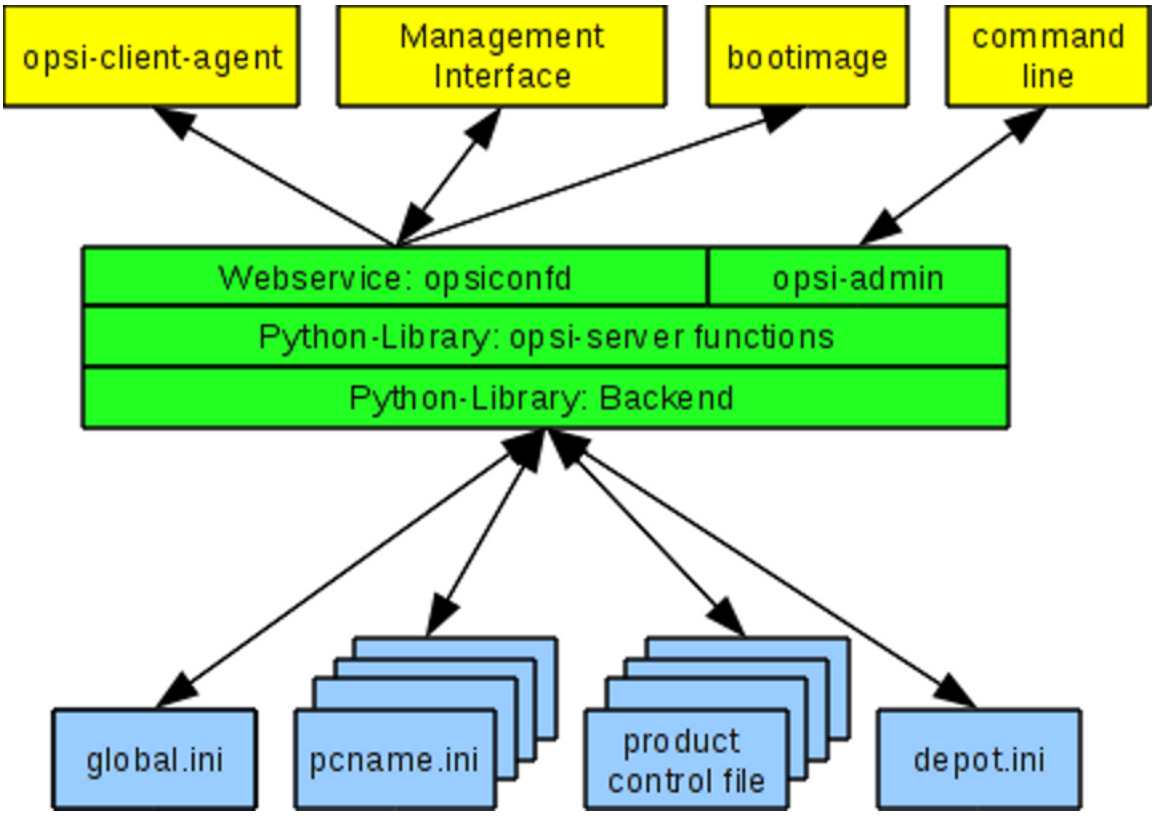


Abbildung 1: Schema: opsi mit File-Backend

Beim MySQL- und LDAP-Backend existieren für die Datentypen jeweils spezifischen Datenobjekte:

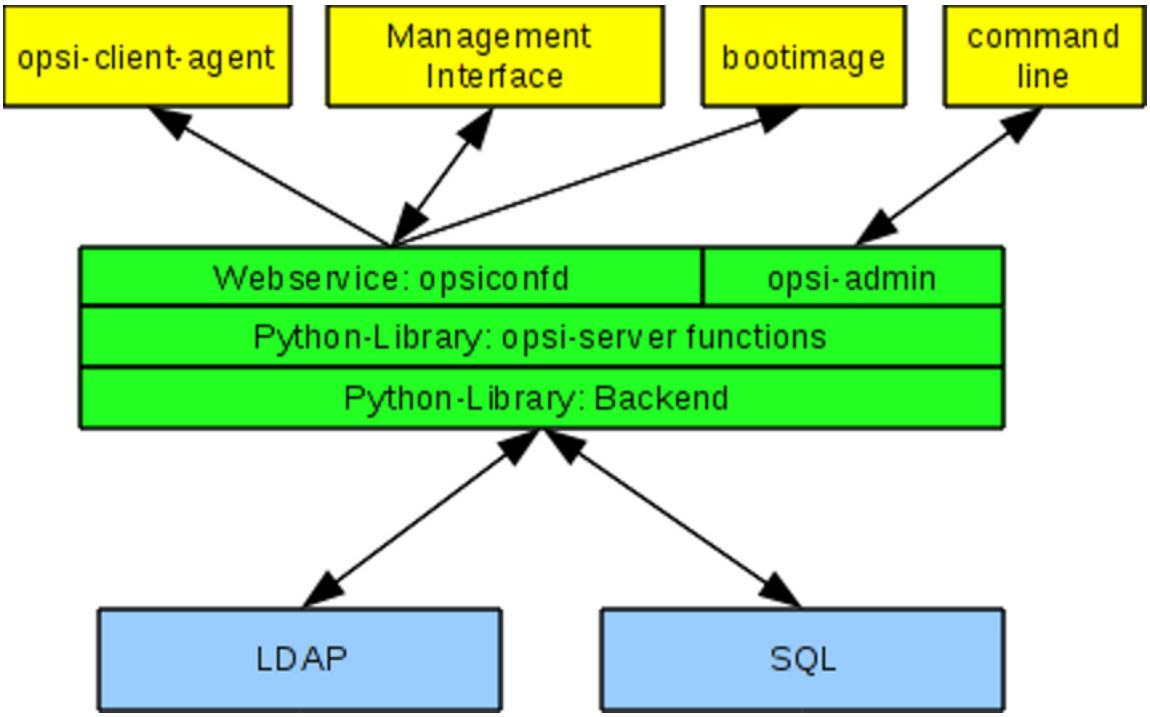


Abbildung 2: Schema: opsi mit SQL- und LDAP-Backend

Mehr zur Datenhaltung finden Sie im Abschnitt 21.

Die Konfiguration der Backends erfolgt in den Konfigurationsdateien in den Verzeichnissen `/etc/opsi/backendManager` sowie `/etc/opsi/backends`.

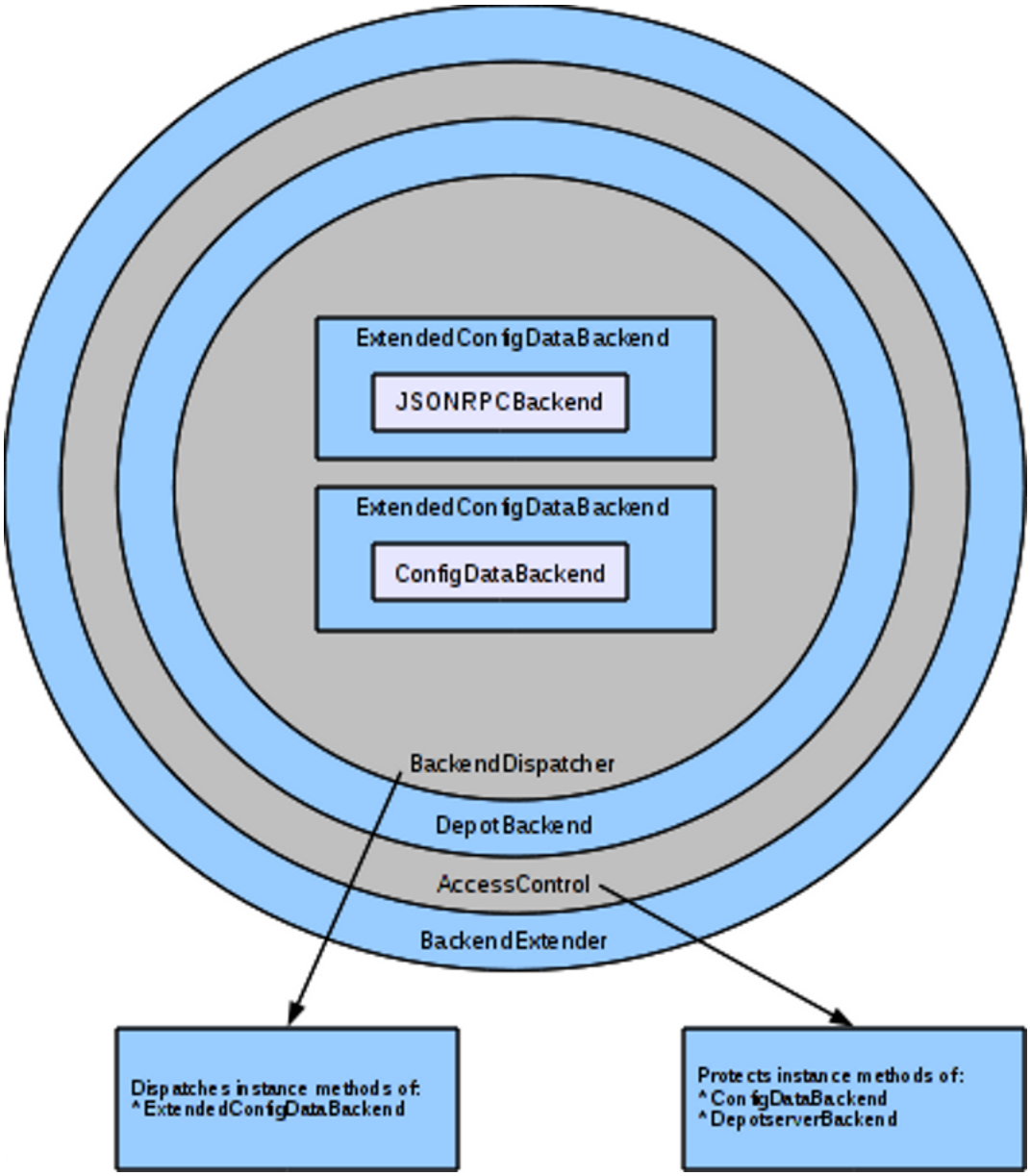


Abbildung 3: Schema: Schichten der Backend-Konfiguration

Die Dateien im opsi3.4-Verzeichnis `/etc/opsi/backendManager.d` spielen keine Rolle mehr.

In den Konfigurationsdateien im Verzeichnis `/etc/opsi/backends` werden die Backends definiert.

Welche Backends für welche Zwecke verwendet werden, steht in der Datei `/etc/opsi/backendManager/dispatch.conf`.

Wer auf welche Methoden zugreifen darf, ist in der Datei `/etc/opsi/backendManager/acl.conf` konfiguriert.

Unterhalb von `/etc/opsi/backendManager/extend.d` können weitere opsi-Methoden definiert sein, welche die Basis opsi-Methoden verwenden.

So ist z.B. das Mapping der "Legacy"-Methoden aus opsi 3 auf die neuen Methoden in der Datei `/etc/opsi/backendManager/extend.d/20_legacy.conf` definiert. Genaue Beschreibungen dieser Konfigurationsdateien finden Sie im Abschnitt 23.1.

4.2 Werkzeug: *opsi-setup*

Das Programm 'opsi-setup` ist das "Schweizer Taschenmesser" zur Einrichtung von opsi.

So greifen zum Beispiel die Pakete zur Installation von opsi auf dem Server auf dieses Skript zurück, um opsi korrekt einzurichten. Da dieses Skript nun auch extern aufrufbar ist, lassen sich damit diverse Wartungsarbeiten und Korrekturen durchführen:

- Registrierung eines opsi-servers als Depotserver
- Verzeichnisrechte korrigieren
- die Backends initialisieren
- Backends upgraden (von 3.4 nach 4.0)
- *mysql-Backend* erstmalig konfigurieren
- Default-Konfigurationen anpassen
- Backends bereinigen
- Samba-Konfiguration anpassen
- DHCP-Konfiguration anpassen

Hier die Hilfe-Anzeige von `opsi-setup`

```
opsi-setup --help

Usage: opsi-setup [options]

Options:
   -h, --help  show this help
   -l          log-level 0..9

   --log-file <path>          path to log file
   --ip-address <ip>          force to this ip address (do not lookup by name)
   --register-depot           register depot at config server
   --set-rights [path]        set default rights on opsi files (in [path] only)
   --init-current-config      init current backend configuration
   --update-mysql             update mysql backend
   --update-ldap              update ldap backend
   --update-file              update file backend
   --configure-mysql          configure mysql backend
   --edit-config-defaults     edit global config defaults
   --cleanup-backend          cleanup backend
   --auto-configure-samba     patch smb.conf
   --auto-configure-dhcpd     patch dhcpd.conf
```

Die Funktionen und Parameter im Einzelnen:

- `--ip-address <ip>`
  Diese Option dient dazu, eine IP-Addresse für den *opsi-server* vorzugeben und damit die Namensauflösung zu umgehen.
- `--register-depot`
  Diese Option dient dazu, einen *opsi-server* an einem anderen *opsi-server* (*opsi-configserver*) als Depotserver anzumelden. Details hierzu vgl. Abschnitt 16.2.
- `--set-rights [path]`
  Setzt bzw. korrigiert die Dateizugriffsrechte in den wesentlichen opsi-Verzeichnissen:
  - `/tftpboot/linux`

  - `/home/opsiproducts`
  - `/var/log/opsi`
  - `/var/lib/opsi`
  - `/opt/pcbin/install`
  - `/etc/opsi`
    Als Parameter kann auch ein Verzeichnis übergeben werden. Dann werden unterhalb dieses Verzeichnisses die Rechte aller opsi-relevanten Verzeichnisse und Dateien gesetzt,
    z.B.:
    `opsi-setup --set-rights /opt/pcbin/install/winxppro/drivers`

- `--init-current-config`
  Initialisiert die eingestellte Backendkonfiguration. Sollte nach jeder Änderung an der `/etc/opsi/backendManager/dispatch.conf` aufgerufen werden.

- Die drei Befehle
  `--update-mysql`
  `--update-ldap`
  `--update-file`
  dienen zum Upgrade des jeweiligen Backends (z.B. von opsi 3.4 auf opsi 4).
  Details siehe *releasenotes-upgrade-Handbuch*.

- `--configure-mysql`
  Dient zur erstmaligen Initialisierung des *mysql-Backend*.
  Details vgl. Abschnitt 21.3.3.

- `--edit-config-defaults`
  Zum Editieren der Defaultwerte der Config wie im *opsi-configed* in der Serverkonfiguration angezeigt.

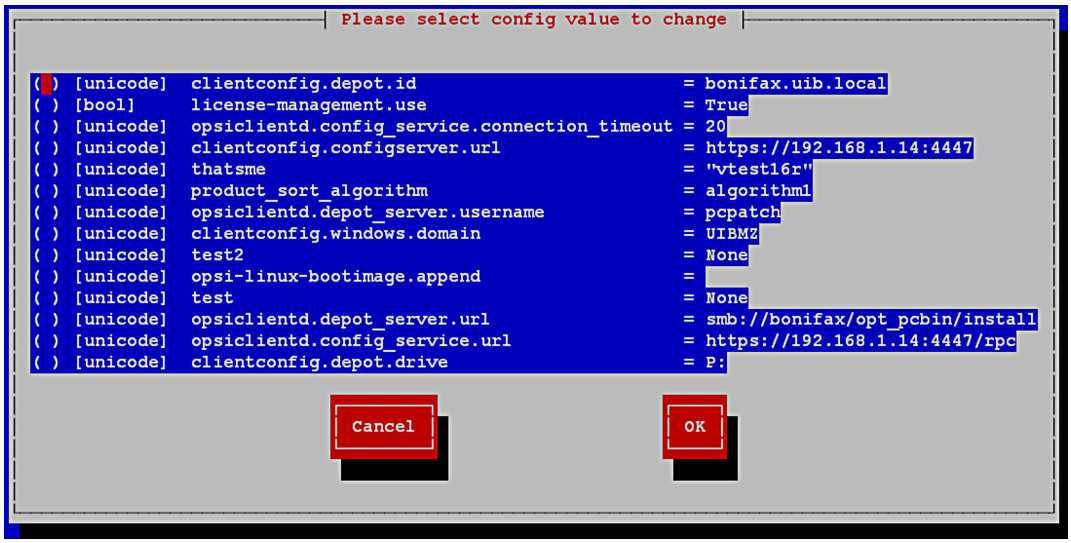


Abbildung 4: Eingabemaske: `opsi-setup --edit-config-defaults`

z.B.:

clientconfig.depot.id
Der Name des Default-Depotservers.

license-management.use
Bestimmt, ob Netboot-Produkte Lizenzkeys aus dem Lizenzmanagement oder aus den Properties holen.

product_sort_algorithm
Bestimmt, nach welchem Algorithmus die Installationsreihenfolge ermittelt wird.

- `--cleanup-backend`
  Überprüft die aktuellen Backends auf Integrität und verwirft unnötige Einträge.
- `--auto-configure-samba`
  Erzeugt in der Samba-Konfigurationsdatei `/etc/samba/smb.conf` die von opsi benötigten shares.
- `--auto-configure-dhcpd`
  Erzeugt in der DHCP-Konfigurationsdatei `/etc/dhcp3/dhcpd.conf` die von opsi benötigten Eintragungen.
  Nur verwenden, wenn der DHCP-Server auf dem opsi Server laufen soll.
  Details hierzu im DHCP-Kapitel des *opsi-getting-started* Handbuchs.

4.3 Werkzeug: Das Managementinterface *opsi-configed*

4.3.1 Voraussetzungen und Aufruf

Der *opsi-configed* setzt Java in der Version 6 voraus und benötigt einen laufenden *opsiconfd* auf der Serverseite.

Wenn *opsi-configed* auf einem Linuxrechner läuft, ist es wichtig darauf zu achten, dass Java 6 in der Sun-Version als Java-Umgebung installiert bzw. konfiguriert ist. Standardmäßig wird bei den Distributionen oft OpenJDK als Javaumgebung installiert. Unter dieser Umgebung kommt es zu subtilen Fehlern beim Configed. Die Lösung ist, ggf. (Sun-) Java nachzuinstallieren und entweder das OpenJDK komplett zu deinstallieren oder (Sun-) Java als Default zu konfigurieren. Letzteres erfolgt auf der Kommandozeile mit dem Befehl

```
update-alternatives -config java
```

Der Aufruf

```
java -version
```

auf der Kommandozeile muss ein Resultat der Form

```
java version "1.6....
Java(TM) SE Runtime Environment ...
```

ergeben.

Meist wird der opsi-configed im Browser aufgerufen per

`https://<servername>:4447/configed`

Der *opsi-configed* ist auch Bestandteil des Clientproduktes *opsi-adminutils* und kann über die entsprechende Gruppe im Startmenü gestartet werden. Serverseitig wird *opsi-configed* als Debianpaket (opsi-configed.xxxxx.deb) installiert und ist über einen Menüeintrag im Desktopmenü sowie über `/usr/bin/opsi-configed` aufrufbar.

Der Aufruf kann auch, wenn das entsprechende Verzeichnis vorgewählt ist, erfolgen über `java -jar configed.jar`.

Der Aufruf `java -jar configed.jar --help` zeigt die Kommandozeilenoptionen:

```
P:\install\opsi-adminutils>java -jar configed.jar --help
starting configed
default charset is windows-1252
server charset is configured as UTF-8

configed [OPTIONS]...

Options:
     -l, --locale    Set locale (format: <language>_<country>)
     -h, --host      Configuration server to connect to
     -u, --user      Username for authentication
     -p, --password  Password for authentication
     -d, --logdirectory Directory for the log files
         --help      Show this text
```

Soll ein anderer Port als der Standardport 4447 verwendet werden, so kann beim Hostnamen der Port mit angegeben werden in der Form: `host:port`.

4.3.2 Login

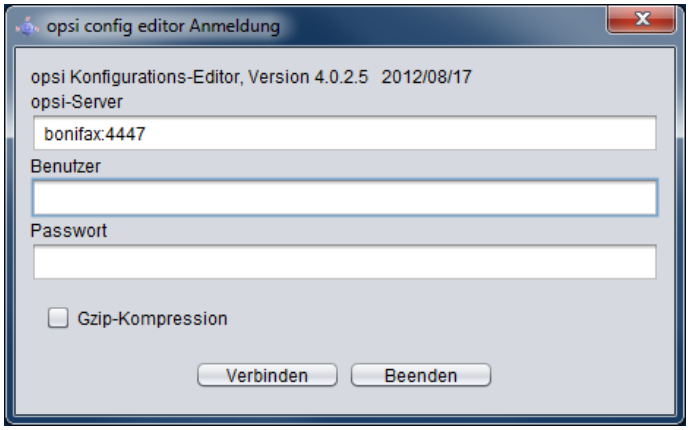


Abbildung 5: Login des *opsi-configed*

Beim Login versucht sich der *opsi-configed* per *https* auf den Port 4447 des *opsiconfd* - bzw. einen anderen angegebene Port - zu verbinden. Das Login geschieht unter Angabe von *Servernamen[:Port]*, eines Usernamens auf dem *opsi-configserver*: und des zugehörigen Unix-Passwortes. Damit das Login erfolgreich ist, muss der User in der Unix-Gruppe *opsiadmin* sein.

4.3.3 Copy & Paste, Drag & Drop

Sie können aus (fast) allen Bereichen des *opsi-configed* die markierten Daten mit den üblichen Tastenkombinationen (*Strg-Insert*, *Strg-C*) in die Zwischenablage kopieren und so anderen Programmen zur Verfügung stellen. Für die meisten Tabellen ist auch die *Drag & Drop*-Funktion aktiviert, mit der die Tabellendaten z.B. nach Excel transferiert werden können.

Achtung
In der Java-Version 1.6.24 hat Oracle das *Copy & Paste* in die System-Zwischenablage aus Security-Gründen deaktiviert, sofern das Programm als Applet im Browser läuft und nicht signiert ist. Standardmäßig wird seit dem configed 4.0.1.11 das Applet mit Signatur ausgeliefert. Damit hat es vollen Systemzugriff.

4.3.4 Client- / Depot-/ Serverkonfiguration / Lizenzmanagement

Um zwischen den verschiedenen Ansichten des opsi-configed zu wechseln, finden sich rechts oben vier Buttons:

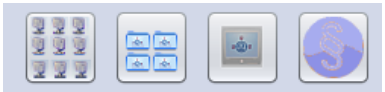


Abbildung 6: opsi-configed: Wahl des Modus

Sie bieten die Funktion an, die Modi Clientkonfiguration, Depotkonfiguration (Depot-Properties) oder Serverkonfiguration (Server-Configs) zu wechseln sowie (in eigenem Fenster) das Lizenzmanagement aufzurufen.

4.3.5 Depotauswahl

Werden an Ihrem *opsi-server* mehrere Depotserver betrieben, so werden diese in einer Liste am linken Rand des `{opsi-configed}` angezeigt.

Per default ist das Depot auf dem *opsi-configserver* markiert und die zu diesem Depot gehörigen Clients werden angezeigt. Sie können mehrere Depots gleichzeitig markieren und deren Clients gemeinsam bearbeiten. - allerdings nur dann, wenn die ausgewählten Depots untereinander synchron sind. Der Versuch, Clients aus asynchronen Depots gemeinsam zu bearbeiten, wird mit einer entsprechenden Warn- und Fehlermeldung abgewiesen.

Nachdem Sie die Auswahl von Depots geändert haben, müssen Sie mit dem Refresh-Button oder mittels des entsprechenden Menüpunkts im Dateimenü "alle Daten neu laden".

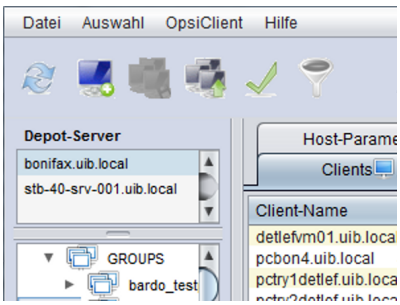


Abbildung 7: *opsi-configed*: Depotauswahl

4.3.6 Clientauswahl und Gruppenbildung

Nach erfolgreichem Login zeigt sich das Hauptfenster mit dem aktiviertem Karteireiter *Clients* zur Auswahl von Clients. Das Hauptfenster zeigt die Liste der Clients, wie sie durch die Depotwahl gegeben bzw. im *Treeview* - s. Abschnitt 4.3.7 - bestimmt ist.

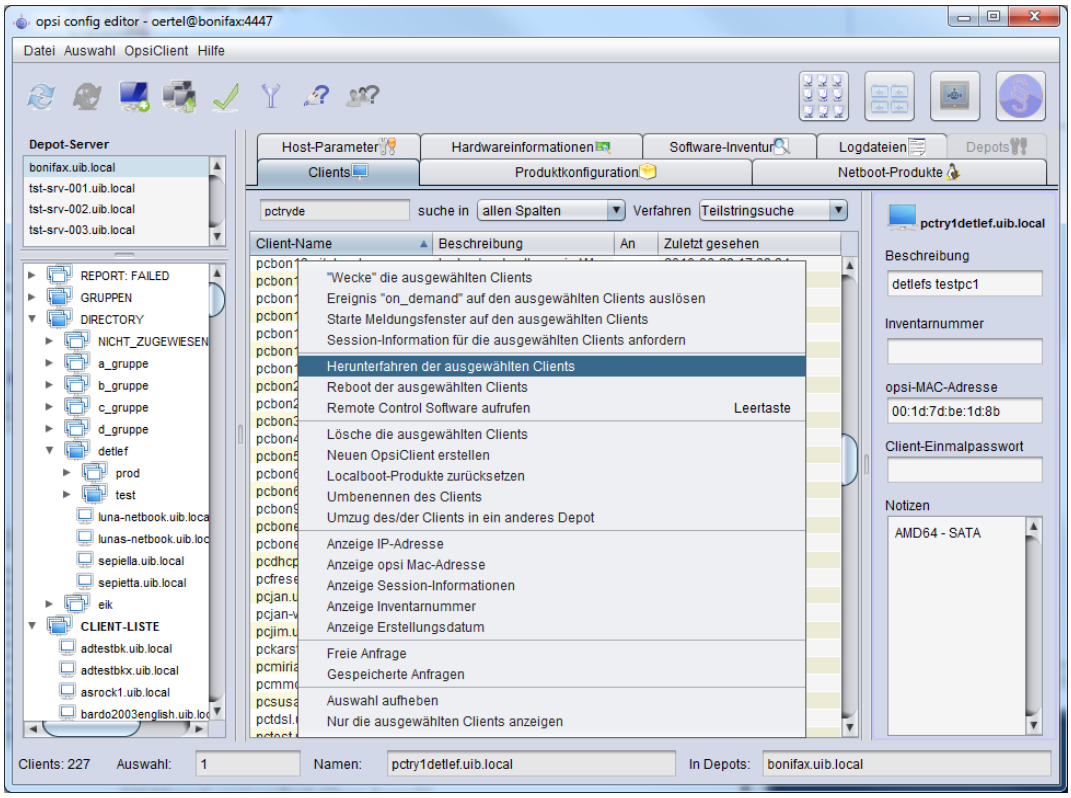


Abbildung 8: *opsi-configed*: Client-Auswahl

In der Liste kann eine Zeile auch über die Suche nach einem Stringwert ausgewählt werden, der im Suchfeld oberhalb der Liste einzugeben ist.

Wie die Suche ausgeführt wird, ist durch die Auswahl in den Drop-down-Listen zu den Feldern und zum Suchverfahren bestimmt. Die Feld-Auswahl bietet an:

- Suche in allen Feldern (genauer, allen Feldern, die nach der momentanten Konfiguration als Spalten dargestellt werden, (Default) oder
- Suche in einem bestimmten Feld.

Beim Suchverfahren kann man sich entscheiden

- ob der Suchstring auch mitten im Text als Teilstring auftauchen darf (Default),
- ob der Feldtext mit dem Suchstring beginnen muss,
- ob der Suchstring als regulärer Ausdruck zum Pattern-Matching dienen soll (für Experten, die reguläre Suche arbeitet mit dem Java-Pattern-Matching).

Betätigen der Return-Taste springt auf den nächsten Treffer der Suche. Weitere Auswahlfunktionen basierend auf der Suche zeigt das Kontextmenü zum Suchfeld.

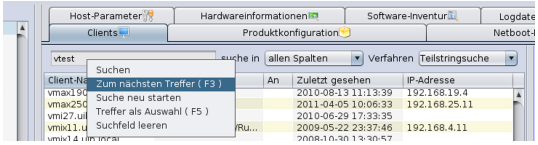


Abbildung 9: *opsi-configed*: Suchfunktion für Client-Auswahl

Die Clientliste

Die tabellarische Auflistung der Clients hat per Default die Spalten *Client-Name*, *Beschreibung*, *An*, *IP-Adresse* und *Zuletzt gesehen*.

- *Client-Name* ist der *full qualified hostname* also der Clientname inklusive (IP-) Domainnamen.
- *Beschreibung* ist eine frei wählbare Beschreibung, die im rechten oberen Teil des Fensters editiert werden kann.
- In der mit *An* betitelten Spalte wird auf Betätigen des Buttons *Prüfen, welche Clients verbunden sind* das Resultat der entsprechenden Anfrage angezeigt.

Abbildung 10: *opsi-configed*: Button *Connected*

- *IP-Adresse* enthält die IP-Nummer, als die der opsi-Server den Clientnamen auflöst.
- *Zuletzt gesehen* gibt Datum und Uhrzeit an, zu der sich der Client zum letzten Mal bei der Softwareverteilung (über den Webservice) gemeldet hat.

In der Defaultkonfiguration deaktiviert sind die Spalten

- *Session-Informationen* (beziehbar vom Betriebssystem des Clients),
- *opsi-Mac-Adresse* (Hardware-Adresse des Clients, die vom Opsi-Server verwendet wird),
- *Inventarnummer* (optional erfasster kennzeichnender String) und
- *Erstellungsdatum* (Datum und Zeit der Client-Erzeugung).

Während einer Sitzung können sie mittels des Kontextmenüs eingeblendet werden. Welche Spalten beim Start des *opsi-configed* angezeigt werden, kann in der Server-Konfiguration mittels des Configs *configed.host_displayfields* bearbeitet werden.

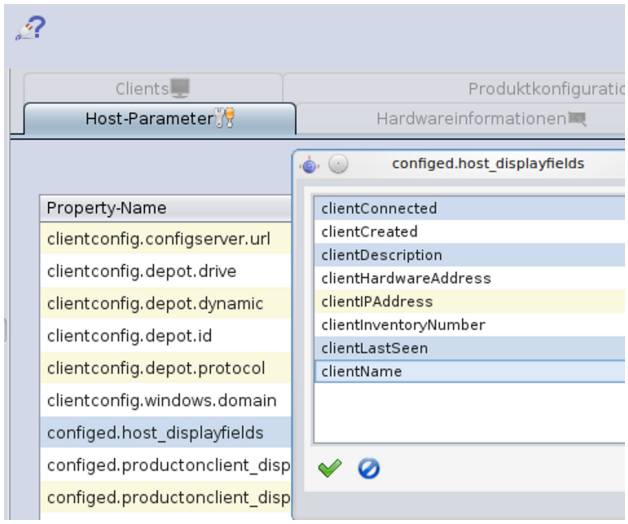


Abbildung 11: *opsi-configed*: Spaltenkonfiguration für die Clientliste

Das Hinzufügen der Spalte *Session-Informationen* schaltet den Button *Abfrage der Session-Informationen von allen Clients* auf aktiv.

Abbildung 12: *opsi-configed*: Button *Sessioninfo*

Bei Betätigen des Buttons versucht sich der *opsiconfd* mit allen Clients zu verbinden und Information über die auf den Clients derzeit aktiven (User-) Sessions einzusammeln. In der Spalte *Session-Informationen* wird der Account-Name der laufenden Sitzungen dargestellt. Mittels Kontextmenü sowie über den Hauptmenüpunkt *OpsiClient* kann das Entsprechende nur für die gerade ausgewählten Clients erreicht werden. Dies vermeidet ggfs. das Warten auf die Netzwerk-Timeouts beim Verbindungsversuch mit gar nicht angeschalteten Rechnern.

Da die Suchfunktion der Clientliste sich (wenn nichts anderes eingestellt ist) über alle angezeigten Spalten erstreckt, kann nach Abruf der Session-Informationen auch nach dem Rechner gesucht werden, auf dem ein User mit bekanntem Account-Namen gerade angemeldet ist.

Die Clientliste kann durch Anklicken eines Spaltentitels nach der entsprechenden Spalte sortiert werden

Auswahl von Clients

In der Clientliste lassen sich ein oder mehrere Clients markieren und so für die (gemeinsame) Bearbeitung auswählen.

Mittels des Trichter-Icons bzw. über *Auswahl / Nur die ausgewählten Clients anzeigen* kann die Anzeige der Clientliste auf die markierten Clients beschränkt werden

Abbildung 13: *opsi-configed*: Trichter-Icon

Die ausgewählten Clients können zu einer existierenden Gruppe hinzugefügt werden (die im Treeview sichtbar ist), indem sie mit der Maus "auf die Gruppe gezogen" werden.

Im Client-Auswahldialog, der über *Auswahl / Auswahl definieren* oder über das Icon *Auswahl definieren* gestartet wird, können dynamische Clientzusammenstellungen anhand wählbarer Kriterien erstellt werden.

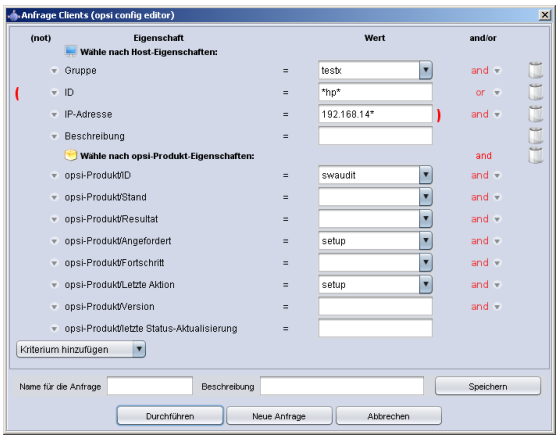


Abbildung 14: *opsi-configed*: Auswahldialog

Neben allgemeinen Eigenschaften der Clients (Rubrik *Wähle nach Host Eigenschaften*) können als Suchkriterium auch (sowohl auf dem PC gefundene als auch mit Opsi installierte) Software sowie Hardware-Komponenten verwendet werden. Bei Texteingaben kann * als Wildcard benutzt werden. Die einzelnen Kriterien können dabei mit logischem "and"(und) bzw. "or"(oder) verknüpft werden, als auch negiert werden durch ein vorangestelltes "not"(nicht).

Die Auswahlliste betitelt *Kriterium hinzufügen* stellt weitere Kriterien zur Definition einer Anfrage bereit. Mit dem Papierkorb-Icon am rechten Rand wird ein Suchkriterium entfernt. Den initialen Zustand der Suchmaske stellt ein Klick auf den Button *Neue Anfrage* wieder her.

Die erstellten Anfragen lassen sich unter einem frei wählbaren Namen speichern. Anschließend kann man sie über *Auswahl / Gespeicherte Suchen...* erneut abrufen. Falls beim Speichern das Feld *Beschreibung* ausgefüllt wurde, wird diese in der Auswahlliste als Tooltip angezeigt.

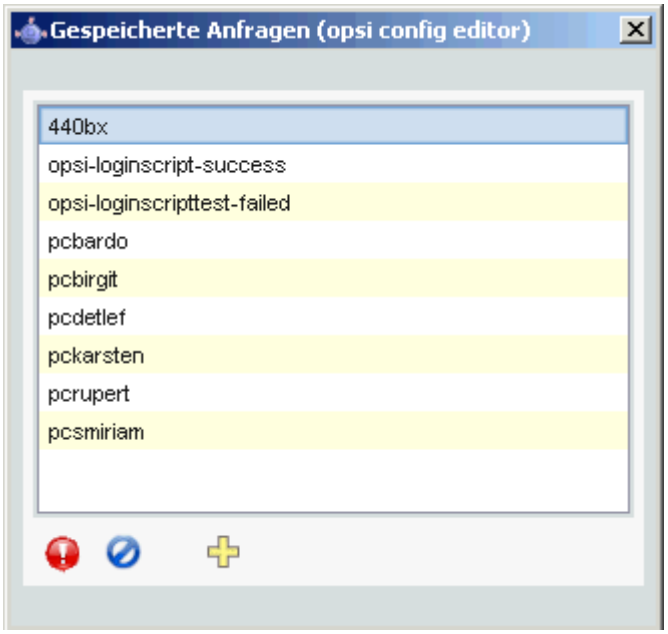


Abbildung 15: *opsi-configed*: Gespeicherte Suchen

Falls bis zum nächsten Aufruf weitere Clients hinzugekommen sind, die den gespeicherten Suchkriterien entsprechen, werden diese beim nächsten Aufruf der Suche ebenfalls gefunden.

Für eine feste Gruppenbildung mit ganz bestimmten Clients siehe weiter unten Abschnitt 4.3.7

Zusätzlich ist noch die Abfrage von Suchen über eine Kommandozeilen-Schnittstelle möglich, um sie auch Skripten zugänglich zu machen. Dazu wird configed mit dem Parameter "-qs" und anschließend dem Namen einer Suche aufgerufen. Lässt man den Namen der Suche weg, wird eine Liste der vorhandenen Suchen ausgegeben.

4.3.7 Clientauswahl und hierarchische Gruppen im *Treeview*

Ab opsi 4.0 lassen sich Clients und Clientgruppen auch mittels eines *Treeview* ansteuern. Diese neue Baumansicht der opsi-Clients ist auf der linken Seite des Hauptfensters des des *opsi-configed* plaziert und bleibt auch beim Wechsel des Tabs sicht- und nutzbar.

Anders als mit der zuvor beschriebenen Speicherfunktion für Client-Sets können mittels der *Treeview*-Komponente auch hierarchische Gruppierungen (also Gruppen in Gruppen) angelegt und verwaltet werden. Dieses Feature ist zunächst ein Ko-Finanzierungsprojekt, welches Sie sich kostenpflichtig (500 €) freischalten lassen können.

Prinzipieller Aufbau

Der Treeview hat einen Basisknoten *ALL*, unterhalb dessen sich zum einen alle Clients für sich befinden, zum anderen der Knoten *GROUPS* als Obergruppe aller selbstdefinierten Gruppen.

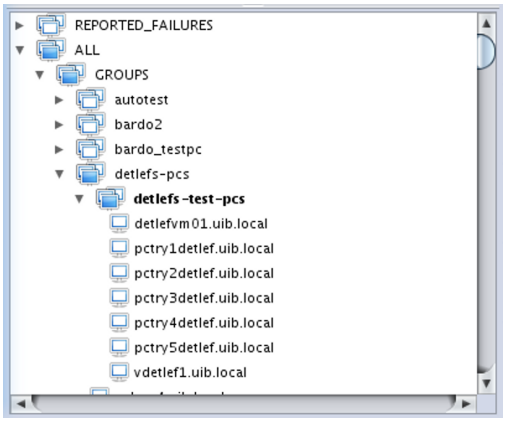


Abbildung 16: *opsi-configed*: *Treeview*

Zusätzlich gibt es den "dynamischen" Knoten *REPORTED_FAILURES*, unterhalb dessen beim Initialisieren des Baums alle Clients (zusätzlich) platziert werden, bei denen bei einer Produktinstallation als Resultat *failed* gespeichert ist.

Festzuhalten ist hier bereits: Zwar hat eine Gruppe einen eindeutigen Ort in der Baum-Hierarchie, aber ein Client kann zu beliebig vielen Gruppen gehören (wobei jede Gruppenmitgliedschaft einen Hinweis auf bestimmte Eigenschaften des Clients darstellen soll).

Wenn ein Client markiert ist, sind alle Gruppen, denen er angehört, mit gefüllter Farbfläche ausgezeichnet.

How to ...

Wenn Sie auf einen Knoten (bzw. eine Gruppe) klicken, so werden alle Clients, die sich unterhalb dieses Knotens befinden, im *Client*-Tab angezeigt (aber kein Client zur Bearbeitung ausgewählt).

Wenn Sie auf einen Client klicken oder im Treeview mehrere Clients per Strg-Klick oder Shift-Klick markieren, so werden diese im Client-Tab angezeigt und zur Bearbeitung markiert.

Sie können auf diese Art und Weise auch den /oder die in Bearbeitung befindlichen Clients wechseln, während Sie in anderen Tabs (wie z.B. Logdateien) arbeiten, ohne zum Client-Tab zurückkehren zu müssen.

In diesem Treeview werden die Gruppen angezeigt, die Sie gemäß Abschnitt 4.3.6 angelegt haben. Sie können zusätzlich Gruppen definieren, indem Sie mit der rechten Maustaste über der Eltern-Gruppe bzw. dem Eltern-Knoten (z.B. Groups) das Kontextmenü öffnen und *Untergruppe erzeugen* wählen.

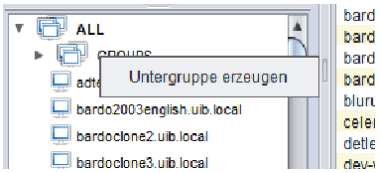


Abbildung 17: *opsi-configed*: Gruppe anlegen

Wenn Sie dies gemacht haben, werden Sie in einem Dialogfenster nach dem Namen der Gruppe gefragt.

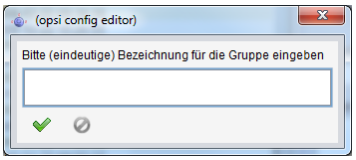


Abbildung 18: *opsi-configed*: Gruppenname eingeben

Eine Gruppe können Sie nun mit Clients füllen, indem Sie per Drag&Drop:

- Clients aus der tabellarischen Clientliste im Client-Tab in die Gruppe kopieren (linke Maustaste)
- Clients aus dem Treeview unterhalb des Knotens *ALL* kopieren (linke Maustaste)
- Clients aus dem Treeview aus anderen Gruppen verschieben (linke Maustaste) oder kopieren (Strg-linke Maustaste)

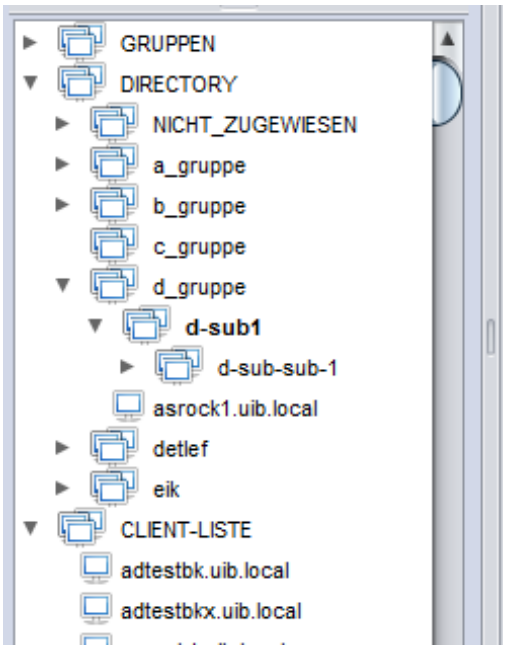


Abbildung 19: *opsi-configed*: *Treeview mit Directory*

Der Bereich *Directory* im Treeview ist ähnlich der von *Gruppen* nur das unterhalb von *Directory* jeder Client exakt an einer Stelle in der Hierarchie auftaucht.

4.3.8 Client-Bearbeitung

Im Client-Tab können Sie über den Menüpunkt *OpsiClient* oder das Kontextmenü eine Reihe clientspezifischer Operationen starten.

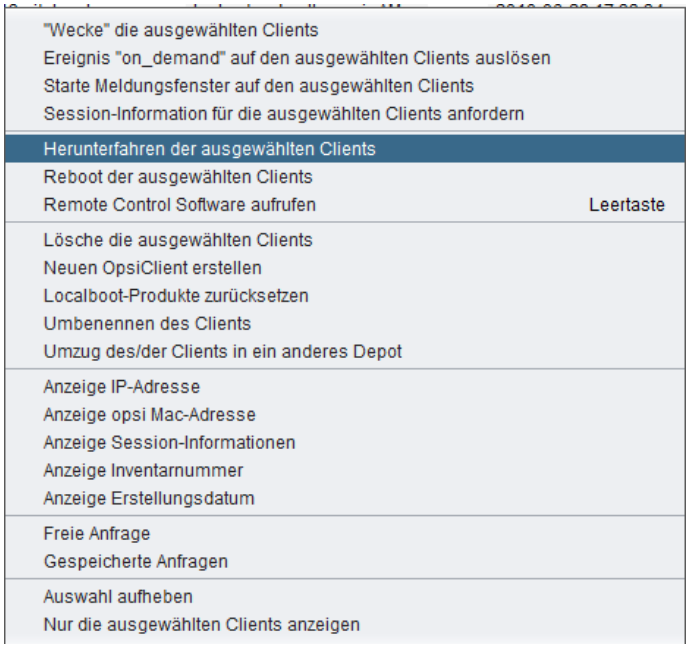


Abbildung 20: *opsi-configed*: Kontextmenü der Clientliste

WakeOnLan

Haben sie einen oder mehrere Clients gewählt, so können Sie diesen über den betreffenden Punkt im *OpsiClient*-Menü bzw. den entsprechenden Eintrag des Kontextmenüs ein *WakeOnLan*-Signal senden.

on_demand Ereignis auslösen (Push Installation)

Über diesen Menüpunkt wird den selektierten Clients ein Aufruf an den *opsi-client-agent* gesendet, dass auf den Clients zu installierende Software jetzt installiert werden soll. Wenn ein Client nicht erreichbar ist, so meldet der *opsi-configed* dies mit einem Fehlerhinweis.

Achtung
Wenn ein Produkt eine Rebootanforderung enthält, so wird der Client ohne Vorwarnung rebootet.

Technisch gesehen wird dem *opsi-client-agent* die Meldung gesendet, er soll das Event *on_demand* auslösen. Das genaue Verhalten ist in der Konfiguration diese Events festgelegt (in der *opsiclientd.conf*).

Nachrichten senden (*Starte Meldungsfenster*)

Über die Auswahl von *Starte Meldungsfenster auf den ausgewählten Clients* erhalten Sie die Möglichkeit, Nachrichten an einen oder mehrere Clients zu senden. Es erscheint zunächst ein Fenster in dem Sie die Nachricht editieren können.

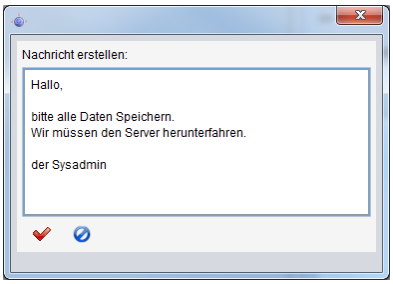


Abbildung 21: *opsi-configed*: Bearbeitungsfenster einer Nachricht

Durch Anklicken des roten Häkchens versenden Sie die Nachricht. Auf den selektierten Clients wird nun die Nachricht angezeigt:

Abbildung 22: *opsi-configed*: Nachrichtenfenster auf dem Client

Sessioninfo der ausgewählten Clients

Sie können den ausgewählten Clients das Signal senden, dem opsi-configed ihre Session-Informationen mit zuteilen. Diese Informationen werden in der entsperchenden Spalte dargestellt, soweit diese eingeblendet ist.

Herunterfahren / Reboot der ausgewählten Clients

Sie können den ausgewählten Clients das Signal senden, herunterzufahren bzw. zu rebooten.

Achtung
Der Client fährt ggf. ohne Rückfrage herunter.

Externe Remotecontrol-Werkzeuge für die ausgewählten Clients aufrufen

Mit der Option *Remote Control Software* im Kontextmenü sowie im Client-Menü (ab *opsi-configed* Version 4.0.1.11) können beliebige Betriebssystem-Kommandos aufgerufen werden, parametrisiert z.B. mit dem Clientnamen.

Beispielhaft sind zwei Konfigurationen zum Anpingen des ausgewählten Hosts automatisch mitgeliefert, und zwar ein Kommando, das unter Windows ausgeführt werden kann, und eines, das in einer graphischen Linux-Umgebung arbeitet. Zur Verdeutlichung: Der *opsi-configed* ruft das jeweilige Kommando aus *seiner* Systemumgebung auf. D.h., die Form des benötigten Kommandos hängt davon ab, ob der *opsi-configed* unter Windows oder Linux läuft.

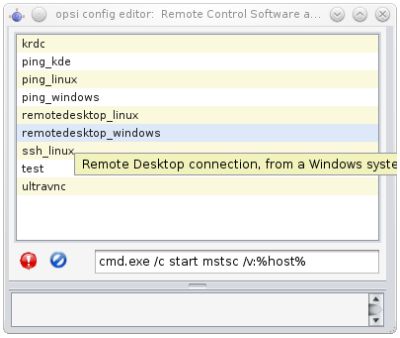


Abbildung 23: *opsi-configed*: Auswahl des Remote-Control-Aufrufs

Das Auswahlfenster ist dreigeteilt: Oben steht die Liste der Namen der verfügbaren Aufrufe. Es folgt eine Zeile, in der das ausgewählte Kommando angezeigt wird sowie, falls zulässig, verändert werden kann. Die Zeile enthält auch Buttons für Start und Abbruch der Aktion. Der dritte Textbereich des Fensters gibt eventuelle Rückmeldungen des Betriebssystems beim Aufruf des Kommandos wieder.

Die Möglichkeiten, die sich bieten, sind quasi unerschöpflich. Z.B. kann ein Kommando konfiguriert werden, das eine RemoteDesktop-Verbindung zum ausgewählten Client öffnet (sofern dieser das zulässt). Unter Windows kann dafür z.B. der folgende Befehl verwendet werden:

```
cmd.exe /c start mstsc /v:%host%
```

Ein entsprechendes Linux-Kommando lautet z.B.:

```
rdesktop -a 16 %host%
```

Dabei ist `%host%` eine Variable, die vom *opsi-configed* automatisch durch den entsprechenden Wert für den Hostnamen ersetzt wird. Analog können auch die Variablen `%ipaddress%` sowie `%inventorynumber%` verwendet werden.

Sofern das Kommando mit *editable* `true` gekennzeichnet ist, können in der angezeigten Kommandozeile z.B. Passwörter oder andere ad-hoc-Variationen des Befehls eingegeben werden.

Sind mehrere Clients markiert, wird das Kommando auf allen ausgewählten parallel ausgeführt.

Die Liste der verfügbaren Kommandos wird über Server-Konfigurationseinträge bearbeitet (vgl. Abschnitt 4.3.15).

Um ein Kommando des Namens `example` zu definieren, muss minimal ein Eintrag `configed.remote_control.example` (oder `configed.remote_control.example.command`) erzeugt werden. Als Wert des Properties wird das Kommando (ggfs. unter Verwendung von Variablen `%host%`, `%ipaddress%` usw.) gesetzt. Zusätzlich kann ein Eintrag der Form `configed.remote_control.example.description` definiert werden. Der Wert dieses Eintrags wird als Tooltip angezeigt. Mit einem Booleschen Eintrag der Form `configed.remote_control.example.editable` kann durch Setzen des Wertes auf `false` festgelegt werden, dass das Kommando beim Aufruf nicht verändert werden darf.

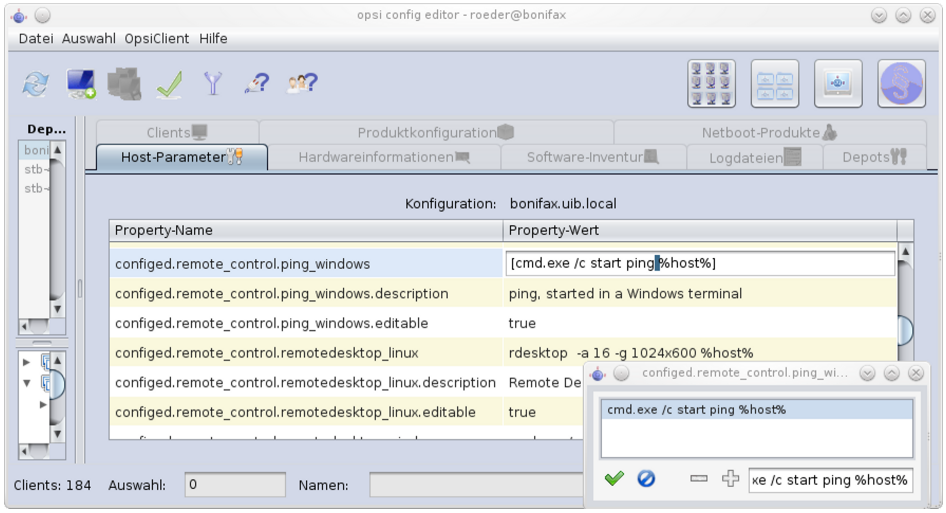


Abbildung 24: *opsi-configed*: Bearbeitung der Remote-Control-Aufrufe bei den Server-Konfigurationseinträgen

Clients entfernen, erstellen, umbenennen, umziehen

Sie können den ausgewählten Clients aus dem opsi-System löschen.

Sie haben hier auch die Möglichkeit, Clients neu anzulegen. Über den Menü-Punkt *Neuen OpsiClient erstellen*, erhalten Sie eine Maske zur Eingabe der nötigen Informationen zur Erstellung eines Clients.

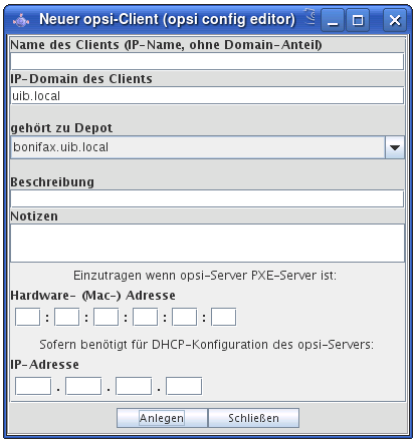


Abbildung 25: *opsi-configed*: Client anlegen

Diese Maske enthält auch Felder für die optionale Angabe der IP-Nummer und der Hardware- (MAC)-Adresse. Wenn das Backend für die Konfiguration eines lokalen DHCP-Servers aktiviert ist (dies ist nicht der Default), werden diese Informationen genutzt, um den neuen Client auch dem DHCP-Server bekannt zu machen. Ansonsten wird die MAC-Adresse im Backend gespeichert und die IP-Nummer verworfen.

Sie können den ausgewählten Client innerhalb von opsi umbenennen. Sie werden dann in einem Dialogfenster nach dem neuen Namen gefragt.

Ein weiterer Dialog steht für den Umzug von einem Client oder mehreren Clients zu einem anderen Depot zur Verfügung:

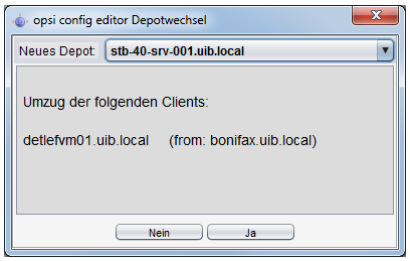


Abbildung 26: *opsi-configed*: Client zu anderem Depot umziehen

Localboot-Produkte zurücksetzen

Sie können alle Informationen zu allen Localbootprodukten der ausgewählten Clients löschen. Dies kann sinnvoll sein um z.B. einen Testclient auf einen definierten Zustand zu setzen.

4.3.9 Produktkonfiguration

Wechseln Sie auf den Karteireiter *Produktkonfiguration*, so erhalten Sie die Liste der zur Softwareverteilung bereitstehenden Produkte und des Installations- und Aktionsstatus zu den ausgewählten Clients.

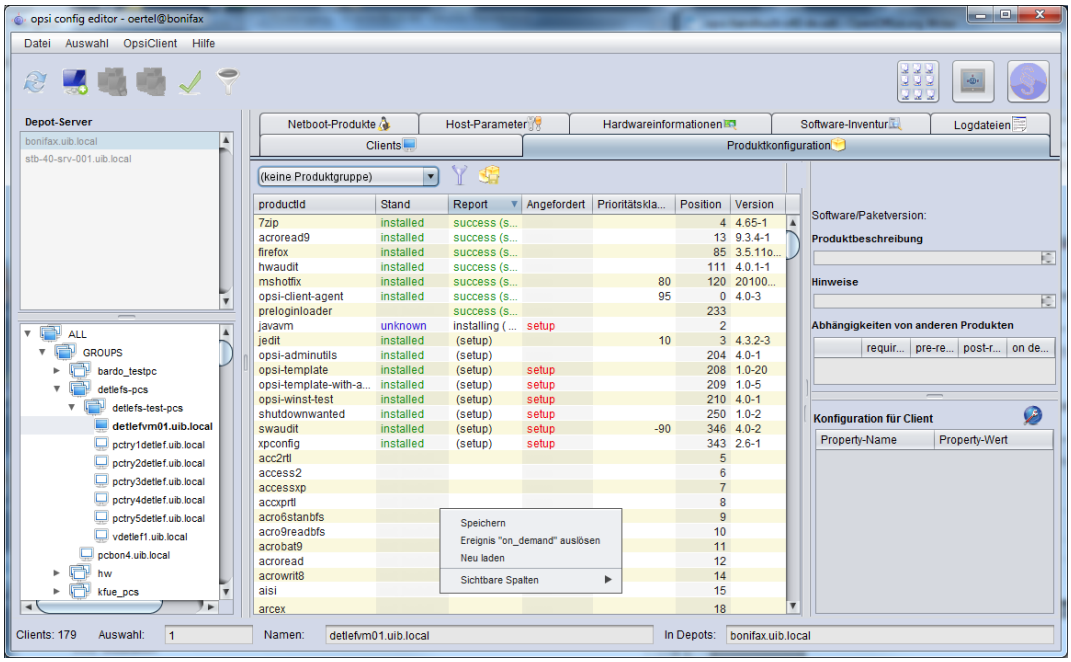


Abbildung 27: *opsi-configed*: Tab Produktkonfiguration

Werte, die für die ausgewählten Clients unterschiedlich sind, werden grau (als Darstellung des Wertes *undefined*) angezeigt.

Sie können auch für die Produktliste eine Sortierung nach einer anderen als der ersten Spalte durch Anklicken des Spaltentitels erreichen. Verfügbar sind die folgenden Spalten:

- *Stand* ist der letzte der Softwareverteilung gemeldete Status zu diesem Produkt und kann die Werte *installed*, *not_installed* und *unknown* haben. *not_installed* wird aus Gründen der Übersichtlichkeit nicht angezeigt. *unknown* ist der Status üblicherweise während einer (De-)Installation sowie wenn das letzte Skript gescheitert ist.

- *Report* ist eine Zusammenfassung der Werte der internen Statusinformationen "Installationsfortschritt" (*actionProgress*), "Ergebnis der letzten Aktion" (*actionResult*) und "letzte angeforderte Aktion" (*lastAction*). Während einer Installation steht hier z.B. *installing* Nach Abschluss einer Aktion enthält das Feld den Text *Ergebnis (letzte Aktion)*, z.B. *failed (setup)* oder *success (uninstall)*.
- *Angefordert* ist die Aktion, welche ausgeführt werden soll. Ein möglicher Wert ist immer *none* (visuell ist das Feld leer). Darüber hinaus sind die Aktionen verfügbar, für die bei diesem Produkt Skripte hinterlegt wurden. Möglich sind *setup*, *uninstall*, *update*, *once*, *always*, *custom*.
- *Prioritätsklasse* gibt an, welche Priorität (100 bis -100) dem Produkt zugeordnet wurde (per default ist die Spalte nicht sichtbar.)
- *Position* gibt an, in welcher Reihenfolge die Produkte installiert werden, per Default ebenfalls keine sichtbare Spalte.
- *Version* ist die Kombination aus Produkt-Version und Package-Version, des auf dem Client installierten opsi-Softwareprodukts.

Durch das Anwählen eines Produkts erhalten Sie auf der rechten Seite des Fensters weitere Informationen zu diesem Produkt. Im Einzelnen:

- *ProduktID:*: Klartextname des Produktes
- *Software/Paketversion*: Produktversion-Paketversion der zur Verteilung bereitstehenden Software (wie sie der Paketierer angegeben hat).
- *Produktbeschreibung*: Freier Text zur im Paket enthaltenen Software.
- *Hinweise*: Freier Text mit Angaben zum Umgang mit diesem Paket.
- *Abhängigkeiten*: Eine Liste von Produkten, zu denen das ausgewählte Produkt Abhängigkeiten aufweist mit Angabe der Art der Abhängigkeit:
  - *required* bedeutet, das ausgewählte Produkt benötigt das hier angezeigte Produkt, es besteht aber keine zwingende Installationsreihenfolge.
  - *pre-required* heißt, das hier angezeigte Produkt muss *vor* dem Ausgewählten installiert werden.
  - *post-required* spezifiziert, das hier angezeigte Produkt muss *nach* dem Ausgewählten installiert werden.
  - *on deinstall* bedeutet, diese Aktion soll bei der Deinstallation des ausgewählten Produktes durchgeführt werden; Reihenfolgen können dabei nicht berücksichtigt werden (weil Installationen und Deinstallationen nicht gemischt vorkommen können, aber ggf. zu entgegengesetzten Reihenfolgeanforderungen führen können).
- *Konfiguration für den Client*: Zur clientspezifischen Anpassung der Installation können für ein Produkt zusätzliche Properties definiert sein. Die Darstellung und Bearbeitung der Tabelle der Properties ist in einem spezifischen Oberflächenelement realisiert:

4.3.10 Property-Tabellen mit Listen-Editierfenstern

Eine Property-Tabelle ist eine zweispaltige Tabelle. In der linken Spalte stehen Property-Namen, denen jeweils in der rechten Spalte ein Property-Wert zugeordnet ist.
Die Zeilen die nicht dem Serverdefault entsprechen werden fett angezeigt.
Rechts oben gibt es zwei Buttons:

- (rechts): Entferne clientspezifische Werte:
  Dieser Button löscht alle Einstellungen beim Client, so dass die Serverdefaults wirken. Werden Serverdefaults geändert wirkt sich dies somit direkt auf den Client aus.
- (links): Setze Client Werte auf Serverdefaults:
  Dieser Button kopiert die Serverdefaults zu den Clienteinstellungen. Damit bleiben diese Einstellungen beim Client erhalten, auch wenn später die Serverdefaults geändert werden.

Sofern entsprechend konfiguriert, wird ein Hinweis zur Bedeutung eines Wertes sowie der Defaultwert angezeigt, sobald der Mauszeiger über die betreffende Zeile bewegt wird ("Tooltip"):

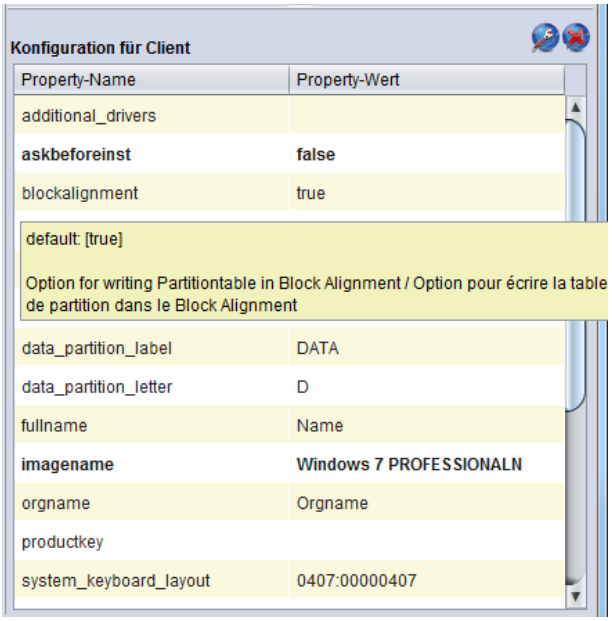


Abbildung 28: *opsi-configed*: Property-Tabelle

Beim Anklicken eines Wertes poppt ein Fenster auf, der *Listen-Editor*, und zeigt einen Wert bzw. eine Liste vorkonfigurierter Werten, wobei der derzeit gültige Wert markiert ist:

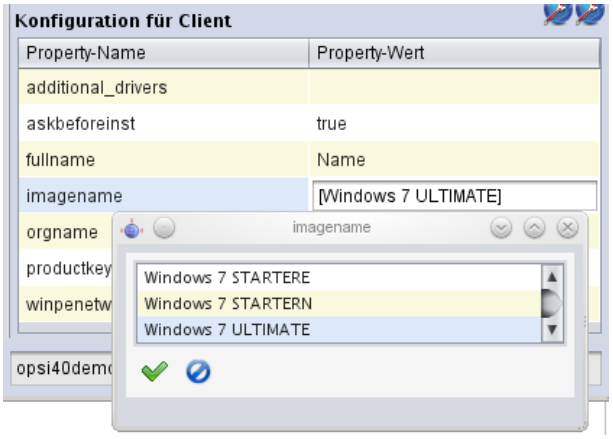


Abbildung 29: *opsi-configed*: Listen-Editor, Auswahlliste

Durch Anklicken eines anderen Wertes kann die Auswahl geändert werden.

Sofern die Liste der zulässigen Werte erweiterbar ist (bzw. die Werte änderbar sind), bietet das Fenster zusätzlich ein Editierfeld an, in dem neue bzw. geänderte Werte eingegeben werden können.

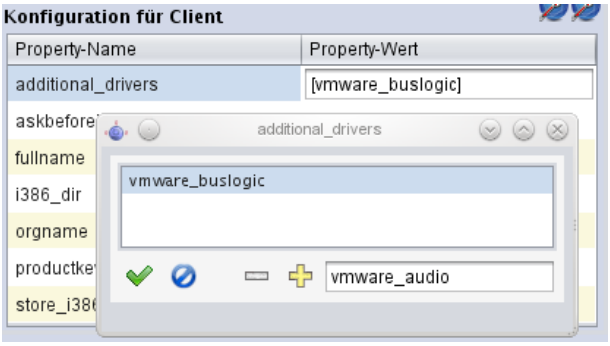


Abbildung 30: *opsi-configed*: Listeneditor, Editierfeld

Zwecks Modifikation kann ein Wert aus der Liste der vorhandenen mit Doppelklick in das Feld übernommen werden. Sobald ein in der Liste noch nicht vorhandener Wert im Editierfeld steht, aktiviert sich das Plus-Symbol zum Übernehmen des neuen Wertes.

Sofern - wie z.B. im Feld *zusätzliche Treiber* der Fall sein sollte - Mehrfach-Werte zulässig sind, erlaubt die Liste eine Mehrfach-Selektion. Wenn die Liste zur Mehrfach-Selektion konfiguriert ist, wird ein Wert mit Strg-Klick zur Selektion hinzugefügt. Mit derselben Tastenkombination lassen sich Werte auch wieder aus der Auswahl entfernen. Das Minus-Symbol leert die Selektion komplett.

Wenn die Liste bearbeitet wurde, wechselt wie auch sonst im configed der grüne Haken nach Rot. Anklicken des Hakens übernimmt den neuen Wert (d.h. die neue Selektion als Wert). Der blaue Cancel-Knopf beendet die Bearbeitung, indem der ursprüngliche Wert zurückgesetzt wird.

4.3.11 Netboot-Produkte

Die Produkte unter dem Karteireiter *Netboot-Produkte* werden analog zum Karteireiter *Produktkonfiguration* angezeigt und konfiguriert.

Die hier angeführten Produkte versuchen, werden sie auf *setup* gestellt, zu den ausgewählten Clients den Start von Bootimages beim nächsten Reboot festzulegen. Dies dient üblicherweise der OS-Installation.

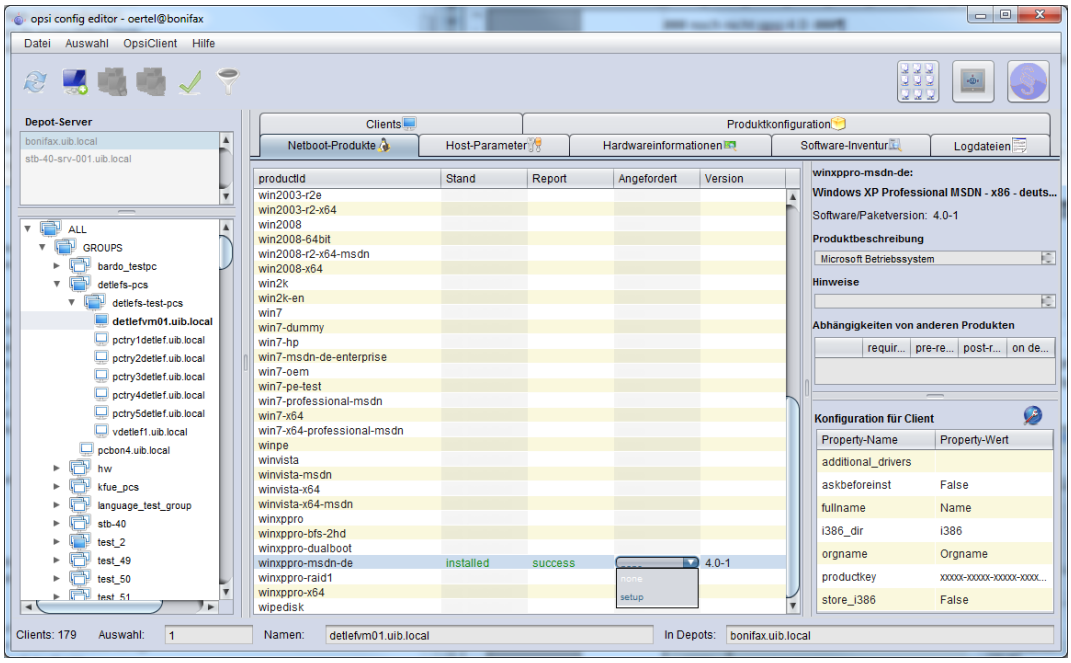


Abbildung 31: *opsi-configed*: Tab Netboot-Produkte

4.3.12 Hardwareinformationen

Unter diesem Karteireiter erhalten Sie die letzten - entweder durch das Bootimage oder durch das Localboot-Produkt hwaudit erhobenen - Hardwareinformationen zum ausgewählten Client.

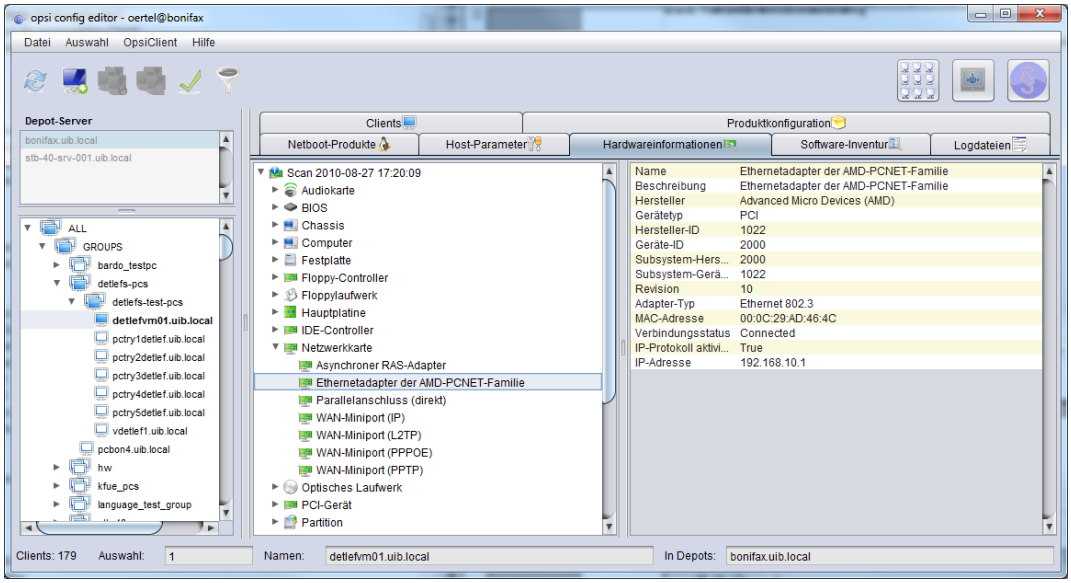


Abbildung 32: *opsi-configed*: Tab Hardware-Information

4.3.13 Software-Inventur

Unter diesem Karteireiter erhalten Sie die letzten mit swaudit ausgelesenen Informationen über installierte Software beim Client.

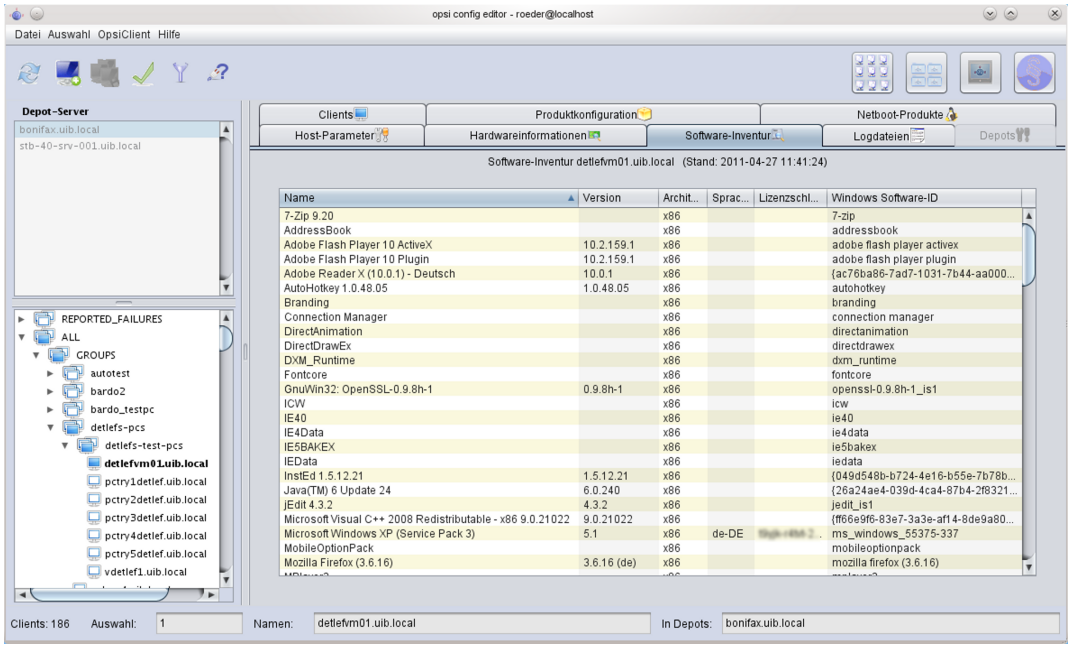


Abbildung 33: *opsi-configed*: Tab Software-Inventur

4.3.14 Logdateien: Logs von Client und Server

Im Kartenreiter Logdateien sind die Logdateien der Clients über den opsi-configed einsehbar. Dabei kann auch in den Logdateien gesucht werden (Fortsetzung der Suche mit *F3* oder *n*. Die einzelnen Zeilen sind je nach Loglevel farbig hervorgehoben.

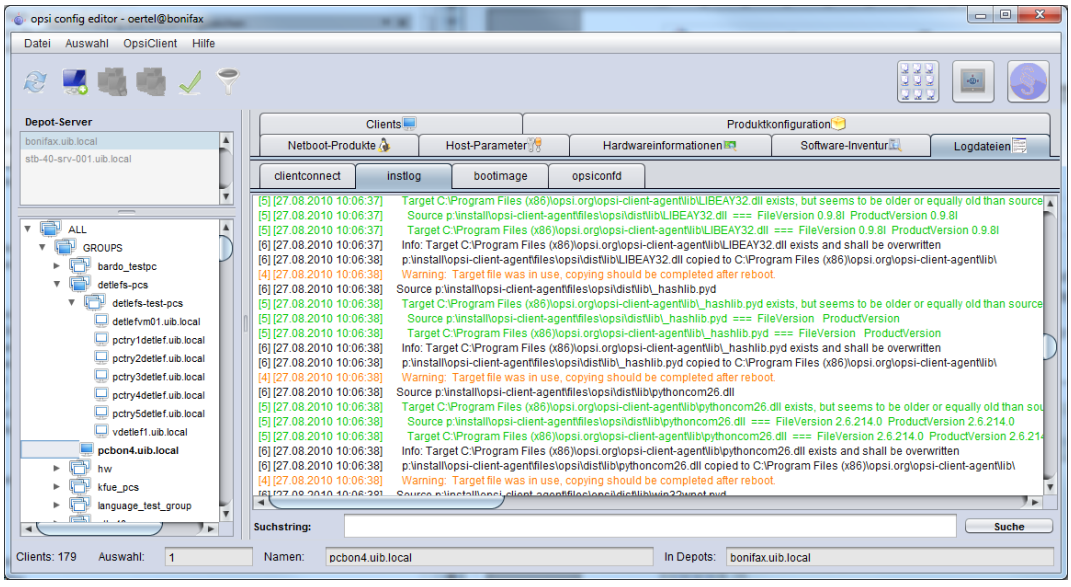


Abbildung 34: *opsi-configed*: Tab Logdateien

4.3.15 Host-Parameter in der Client- und der Serverkonfiguration

Einige Konfigurationsdaten können Sie über den Tab *Host-Parameter* setzen und zwar als Server-Defaults im Modus Serverkonfiguration (vgl. Abschnitt 4.3.4) und clientspezifisch in der Clientkonfiguration.

Konfigurationseinträge (config-Objekte des opsi-Servers) sind grundsätzlich Wertelisten. Als Werkzeug zur Bearbeitung der Werte dient daher der Listeneditor (vgl. Abschnitt 4.3.10)

Je nach Definition des jeweiligen Konfigurationsobjekts können die Werte der Liste

- Unicode-Textwerte oder boolesche Werte (`true`/`false`) sein,
- die Liste kann ein- oder mehrelementig sein
- und die Auswahl für die Listenelemente kann fest oder erweiterbar sein.

Neue Konfigurationseinträge der Typen Unicode-Liste (erweiterbar) sowie boolesche Liste (fix) können über das Kontextmenü erstellt werden. Ebenso können bestehende Einträge gelöscht werden.

Das Verhältnis von Server- und Client-Hosteinträgen ist verwickelt:

- Server-Einträge liefern die Default-Werte für die Client-Einträge.
- Wenn ein Server-Eintrag (das config-Objekt) gelöscht wird, verschwinden auch die zugehörigen Client-Einträge.
- Wenn ein Client-Eintrag im *opsi-configed* neu angelegt wird, wird automatisch ein passendes Server-Konfigurationsobjekt erzeugt.
- Wenn ein Client-Eintrag im *opsi-configed* gelöscht wird, wird nur ein eventuell vorhandener spezifischer Wert gelöscht und wieder der Server-Default verwendet.
- Wenn ein spezifischer Client-Wert sich vom Server-Default unterscheidet, so ist dieser fett dargestellt.
- Für bestimmte Konfigurationsobjekte können zwar Client-Werte erzeugt und bearbeitet werden, sie haben jedoch derzeit keinerlei Funktion (z.B. die Einträge für den *opsi-configed*, die mit `configed.` beginnen).

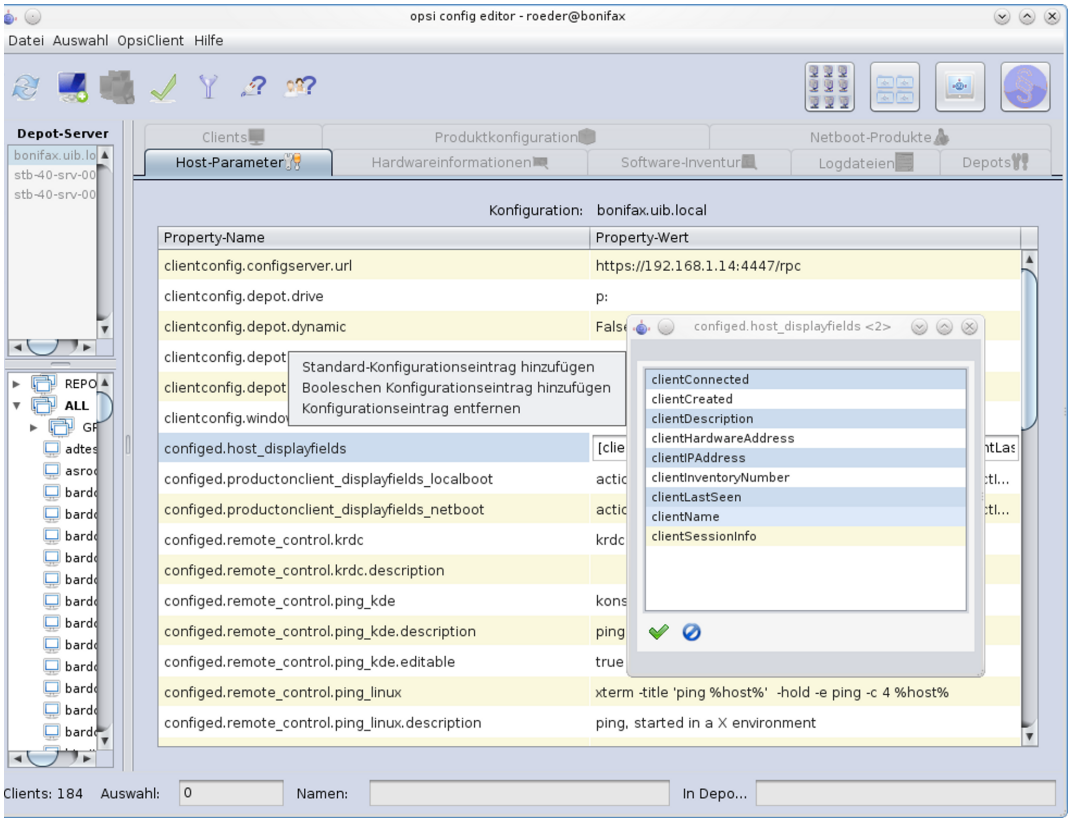


Abbildung 35: *opsi-configed*: Tab Hostparameter (Server- und Client-Konfiguration)

4.3.16 Depotkonfiguration

Im Modus Depotkonfiguration (vgl. Abschnitt 4.3.4) öffnet sich der Tab *Depots*. Nach Auswahl eines Depotservers in der Drop-Down-Liste können Werte bearbeitet werden, die das Depot parametrisieren.

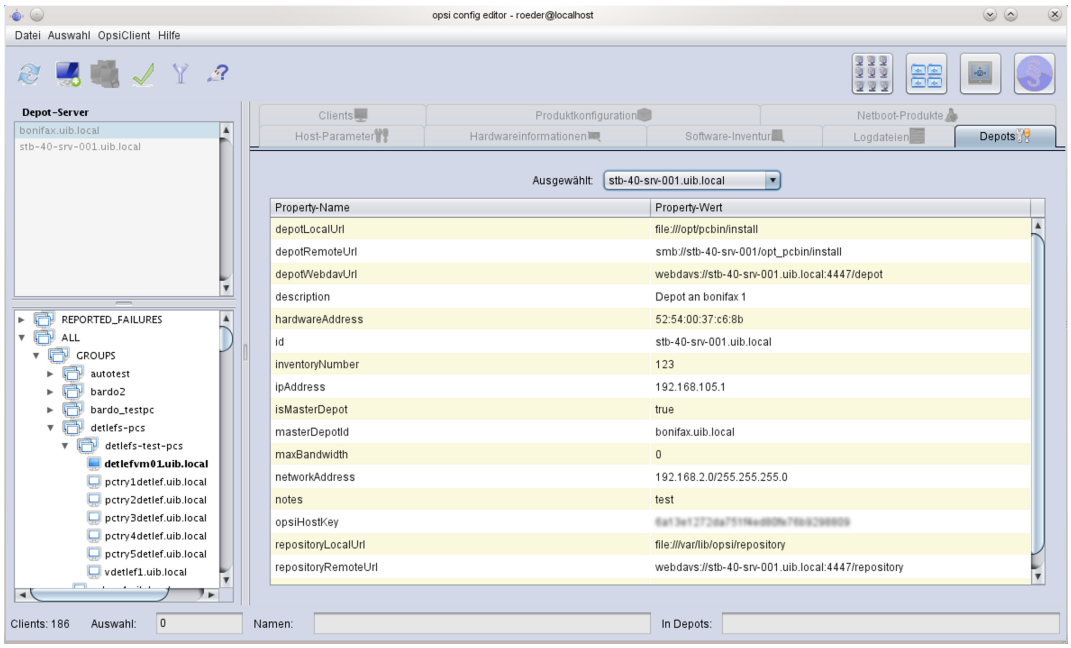


Abbildung 36: *opsi-configed*: Tab Depot-Konfiguration

4.4 Werkzeug *opsi-package-manager*: opsi-Pakete (de-) installieren

Der *opsi-package-manager* dient zur (De-) Installation von Produkt-Paketen auf einem opsi-server.

Beim Aufruf von *opsi-package-manager* zur Installation muss das zu installierende Paket für den Systemuser *opsiconfd* lesbar sein. Es wird daher dringend empfohlen, Produkt-Pakete aus */home/opsiproducts* bzw. einem Unterverzeichnis hiervon zu installieren.

Die Logdatei des *opsi-package-managers* ist */var/log/opsi/package.log*

Paket installieren (ohne Fragen neu installieren):

```
opsi-package-manager -i softprod_1.0-5.opsi'
```

Paket installieren (mit Fragen nach Properties):

```
opsi-package-manager -p ask -i softprod_1.0-5.opsi
```

Paket installieren (und für alle auf Setup stellen, bei denen es installiert ist):

```
opsi-package-manager -S -i softprod_1.0-5.opsi
```

Paket deinstallieren:

```
opsi-package-manager -r softprod
```

Paket extrahieren und umbenennen:

```
opsi-package-manager -x opsi-template_<version>.opsi --new-product-id myprod
```

Eine Übersicht über alle Optionen liefert die Option --help.

Zu beachten:

- Die Optionen -d bzw. --depots sind für den multi-depotserver Betrieb und werden nur mit einem entsprechenden Supportvertrag unterstützt.
- Bei der Verwendung von -d wird das zu installierende Paket zunächst nach `/var/lib/opsi/repository` kopiert. Dort muss ausreichend Platz zur Verfügung stehen.
  Siehe hierzu auch: opsi-server mit mehreren Depots Abschnitt 16

```
#opsi-package-manager --help

usage: opsi-package-manager [options] <command>

Manage opsi packages

Commands:
  -i, --install      <opsi-package> ...      install opsi packages
  -u, --upload       <opsi-package> ...      upload opsi packages to repositories
  -l, --list         <regex>                 list opsi packages matching regex
  -D, --differences  <regex>                 show depot differences of opsi packages matching regex
  -r, --remove       <opsi-product-id> ...   uninstall opsi packages
  -x, --extract      <opsi-package> ...      extract opsi packages to local directory
  -V, --version                              show program's version info and exit
  -h, --help                                 show this help message and exit

Options:
  -v, --verbose                   increase verbosity (can be used multiple times)
  -q, --quiet                     do not display any messages
  --log-file         <log-file>   path to debug log file
  -d, --depots       <depots>     comma separated list of depot ids to process
                                     all = all known depots
  -p, --properties   <mode>       mode for default product property values
                                     ask     = display dialog
                                     package = use defaults from package
                                     keep    = keep depot defaults (default)
  --purge-client-properties       remove product property states of the installed product(s)
  -f, --force                     force install/uninstall (use with extreme caution)
  -U, --update                    set action "update" on hosts where installation status is "installed"
  -S, --setup                     set action "setup" on hosts where installation status is "installed"
  -o, --overwrite                 overwrite existing package on upload even if size matches
  -k, --keep-files                do not delete client data dir on uninstall
  -t, --temp-dir     <path>       tempory directory for package install
  --max-transfers    <num>        maximum number of simultaneous uploads
                                     0 = unlimited (default)
  --max-bandwidth    <kbps>       maximum transfer rate for each transfer (in kilobytes per second)
                                     0 = unlimited (default)
  --new-product-id   <product-id> set a new product id when extracting opsi package
```

4.5 Werkzeug: *opsi-product-updater*

Das Kommandozeilen Werkzeug `opsi-product-updater` dient dazu, komfortabel opsi-Produkte aus einem Repository zu laden und auf dem Server zu installieren. Daneben kann es auch per cronjob zeitgesteuert aufgerufen werden und so zur automatischen Synchronisation von opsi-Servern bzw. für automatische Updates verwendet werden.

```
# opsi-product-updater --help

Usage: opsi-product-updater [options]
Options:
    -h    Show this help text
    -v    Increase verbosity (can be used multiple times)
    -V    Show version information and exit
    -c    Location of config file
```

Die wesentlichen Features sind *konfigurierbare Repositories* und *konfigurierbare Aktionen* (die Konfigurationseinstellungen werden in der `/etc/opsi/opsi-product-updater.conf` vorgenommen).

4.5.1 Konfigurierbare Repositories

Repositories sind die Quellen, von denen sich der opsi-server die Pakete holt.

Grundsätzlich gibt es zwei Arten von Repositories, *Internet-Repositories* und *opsi-Server*:

Internet-Repositories

Das wichtigste Beispiel ist das uib-Repository mit der URL http://download.uib.de

Internet-Repositories sind gekennzeichnet durch die Parameter

- *baseURL* (z.B. http://download.uib.de)
- *dirs* (Eine Liste von Verzeichnissen z.B. opsi4.0/produkte/essential)
- sowie bei Bedarf *username* und *password* für Passwort-geschützte Repositories (z.B. für Abo-Kunden)

Bei Bedarf ist auch ein Proxy einzustellen.

opsi-server

Ein Repository hat den Typ *opsi-server*, wenn in der `/etc/opsi/opsi-product-updater.conf` in der Sektion des Repositorys ein Eintrag zum Punkt

- *opsiDepotId*

vorgenommen wird.

In der Regel ist bei einem *opsi-depotserver* an diese Steller der zentrale *configserver* einzutragen. Damit zieht der Depotserver seine Pakete per Aufruf des *opsi-product-updater* bzw. automatisiert per Cronjob vom zentralen Server.

4.5.2 Konfigurierbare Aktionen

Für jedes Repository kann eingestellt werden:

- *autoupdate*: Aktuellere Versionen installierter Pakete werden geholt und installiert.
- *autoinstall*: Auch bis jetzt nicht installierte Pakete werden geholt und installiert
- *autosetup*: Die geholten und installierten Pakete werden für alle Clients, auf denen dieses Produkt installiert ist, auf *setup* gesetzt.

Zusätzlich ist es möglich, die Aktualisierung der Pakete auf den Clients über einen konfigurierbaren Wake-On-Lan-Mechanismus anzustoßen. In Verbindung mit dem Produkt *shutdownwanted* kann dafür gesorgt werden, dass die Clients nacheinander geweckt, die Software verteilt und die Clients danach wieder heruntergefahren werden. Hierdurch kann man seine Clients zum Beispiel außerhalb der Geschäftszeiten mit Updates und Software versorgen und die Anwender können am nächsten Morgen direkt mit der Arbeit beginnen.

4.6 Werkzeuge: opsi-admin / *opsi config interface*

4.6.1 Übersicht

Seit opsi 3.0 enthält eine serverseitige Bibliothek die zentralen Zugriffsfunktionen auf die opsi-Datenhaltung. Nach außen bietet sie eine API an, mit der ihre Funktionen genutzt werden können. Der *opsiconfd* stellt die komplette API als Webservice zur Verfügung.

Über den Aufruf von https://<opsi-server>:4447/interface kann über ein grafisches Frontend in elementare Form auf diesen Webservice zugegriffen werden. Dazu müssen Sie sich als Mitglied der Gruppe *opsiadmin* authentifizieren.

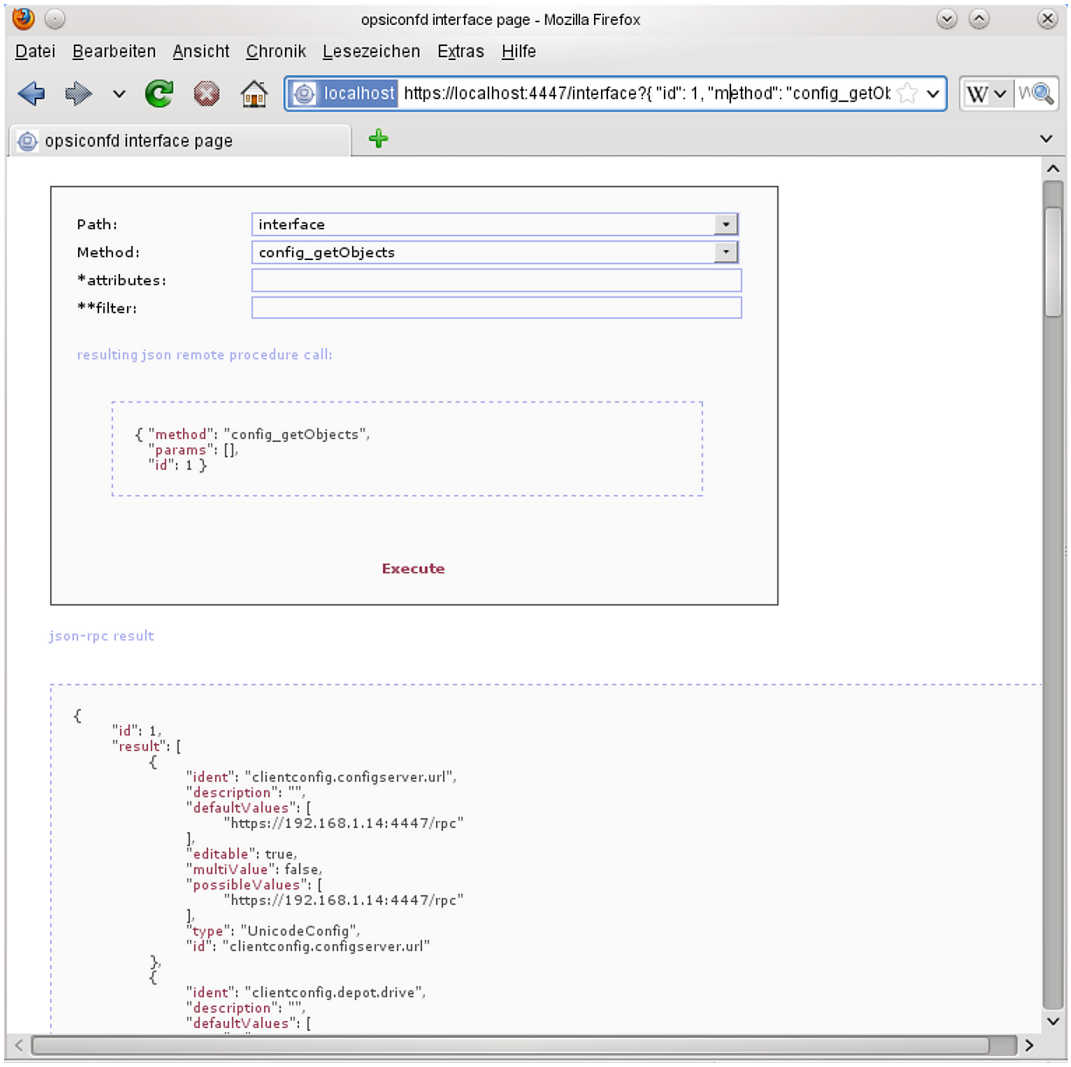


Abbildung 37: *opsiconfd*: Web-Interface

Auf der Kommandozeile kann mit dem Befehl `opsi-admin` auf die API zugegriffen werden. Dabei bietet *opsi-admin* einen interaktiven Modus und einen nicht-interaktiven z.B. zum Einsatz in Skripten.

Der Aufruf von `opsi-admin --help` zeigt eine kleine Hilfe zu den Optionen:

```
# opsi-admin --help

Usage: opsi-admin [options] [command] [args...]
Options:
  -h, --help           Display this text
```

```
  -V, --version        Display this text
  -u, --username       Username (default: current user)
  -p, --password       Password (default: prompt for password)
  -a, --address        URL of opsiconfd (default: https://localhost:4447/rpc)
  -d, --direct         Do not use opsiconfd
      --no-depot       Do not use depotserver backend
  -l, --loglevel       Set log level (default: 3)
                       0=nothing, 1=essential, 2=critical, 3=error, 4=warning
                       5=notice, 6=info, 7=debug, 8=debug2, 9=confidential
  -f, --log-file       Path to log file
  -i, --interactive    Start in interactive mode
  -c, --colorize       Colorize output
  -S, --simple-output  Simple output (only for scalars, lists)
  -s, --shell-output   Shell output
```

opsi-admin kann auf einen opsi-Webservice zugreifen oder direkt auf der Datenhaltung arbeiten. Für die Arbeit über den Webservice müssen neben der URL auch *username* und *password* angegeben werden. Dies wird man in Skripten üblicherweise nicht tun wollen. Stattdessen bietet sich hier der direkte Datenzugriff über Aufruf `opsi-admin -d` an.

Im interaktiven Modus (Start mit `opsi-admin -i` bzw. `opsi-admin -d -i -c`, kurz `opsi-admin -dic`) erhalten Sie Eingabe-Unterstützung durch die Tabtaste. Betätigen der Tabtaste führt auf eine Auswahl der der möglichen Fortsetzungen der Eingabe bzw. die Angabe des Datentyps der nächsten erwarteten Eingabe. In der Liste der möglichen Eingaben können Sie mit Bild-auf und Bild-ab blättern.

Die Optionen `-s` und `-S `erzeugen eine Form der Ausgabe welche sich leichter in Skripten weiterverarbeiten lässt.

Außer den Methodenaufrufen (eingeleitet mit `method`), welche direkt die API widerspiegeln, gibt es Aufrufe (eingeleitet mit `task`), die intern auf eine Kombination von Methodenaufrufen zur Erledigung einer bestimmten Aufgabe abgebildet werden.

4.6.2 Typische Verwendung

Ein Produkt für alle Clients auf setup stellen, welche dieses Produkt installiert haben:

```
opsi-admin -d task setupWhereInstalled "softprod"
```

Liste aller Clients

```
opsi-admin -d method host_getIdents
```

Client löschen

```
opsi-admin -d method host_delete <clientname>
```

z.B.:

```
opsi-admin -d method host_delete "pxevm.uib.local"
```

Client anlegen

```
opsi-admin -d method host_createOpsiClient <full qualified clientname>
```

z.B.:

```
opsi-admin -d method host_createOpsiClient "pxevm.uib.local"
```

Action request setzen

```
opsi-admin -d method setProductActionRequest <productId> <clientId> <actionRequest>
```

z.B.:

```
opsi-admin -d method setProductActionRequest win7 pxevm setup
```

Beschreibungen den Clients zuordnen

```
opsi-admin -d method setHostDescription "dpvm02.uib.local" , "Client unter VMware"
```

Pcpatch-Passwort setzen

```
opsi-admin -d task setPcpatchPassword
```

Setzt das Passwort von pcpatch für Unix, samba und opsi.

4.7 Serverprozesse: opsiconfd und opsipxeconfd

Der *opsipxeconfd* dient zur Bereitstellung von *named pipes* im `tftpboot`-Bereich, welche den Bootvorgang eines PCs über das PXE-Protokoll steuern.

Die zugehörige Konfigurationsdatei ist `/etc/opsi/opsipxeconfd.conf`, die Logdatei `/var/log/opsi/opsipxeconfd.log`.

Der *opsiconfd* dient zur Bereitstellung der opsi-server-API als JSON-Webservice und nimmt noch eine Reihe weiterer Aufgaben wahr.

Dieser Dienst ist damit der zentrale opsi-Dienst. Über ihn wird z.B. sämtliche Kommunikation zwischen den Clients und dem Server abgewickelt.

Von daher ist die Möglichkeit, diesen Prozess und seine Last zu überwachen, ein wichtiges Werkzeug.

4.7.1 *opsiconfd*-Überwachung: opsiconfd info

Unter der Webadresse https://<opsi-server>:4447/info erhalten Sie grafisch aufbereitete Informationen über den Lastverlauf des *opsiconfd* der letzten Stunde, des letzten Tages, des letzten Monats und des letzten Jahrs sowie weitere tabellarische Informationen.

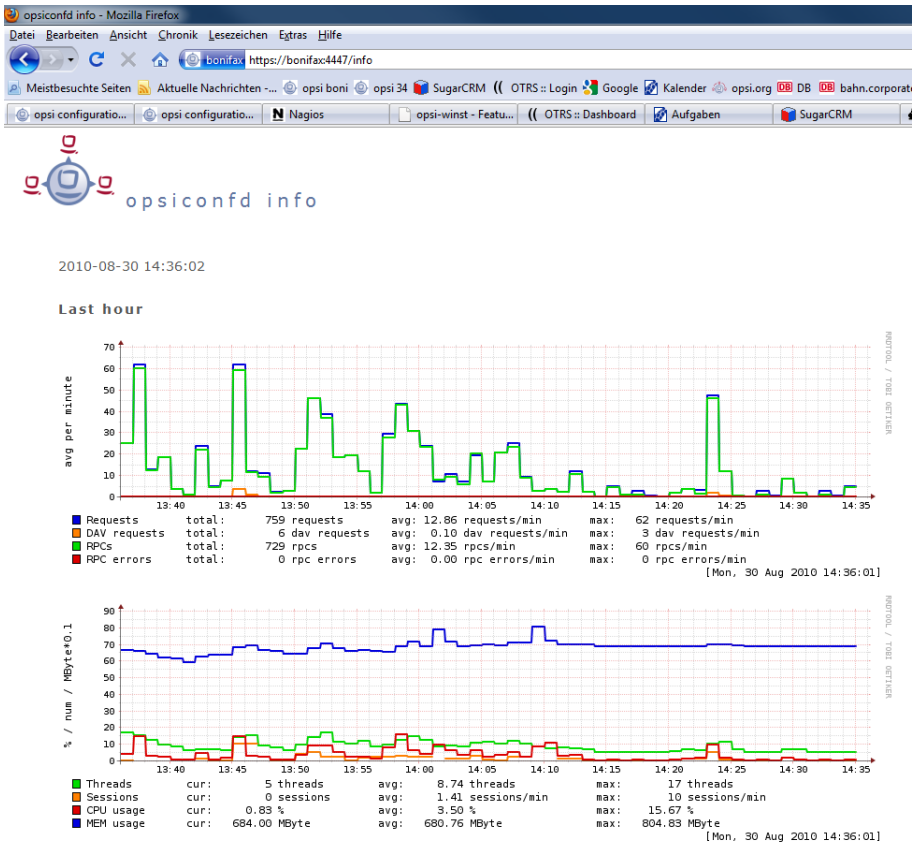


Abbildung 38: opsiconfd info: *opsiconfd*-Werte der letzten Stunde

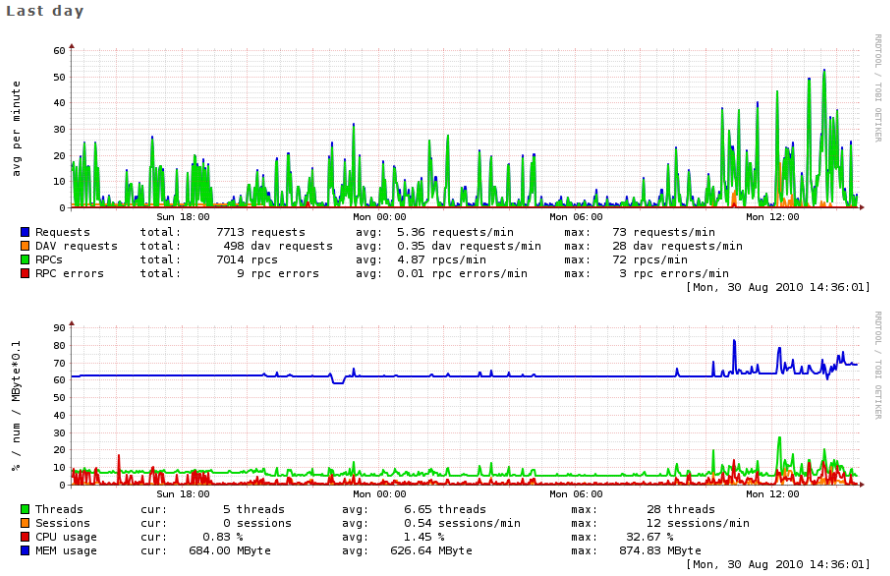


Abbildung 39: opsiconfd info: *opsiconfd*-Werte des letzten Tages

5 Web service / API Methoden

5.1 Web service / API Methoden seit opsi 4.0

5.1.1 Übersicht

Für opsi 4.0 wurden die Backends und die dazugehörigen Methoden komplett überarbeitet.

Die opsi4-Backends basieren auf Objekten. Ein Objekt hat eine Reihe von Eigenschaften.

Als Beispiel diene hier das Objekt *product*. Das Objekt vom Typ *product*, welches das opsi-Produkt *javavm* beschreibt sieht z.B. so aus:

```
"ident": "javavm;1.6.0.20;2"
"id": "javavm"
"description": "Java 1.6"
"changelog": ""
"advice": ""
"userLoginScript": ""
"name": "SunJavaRuntimeEnvironment"
"priority": 0
"packageVersion": "2"
"productVersion": "1.6.0.20"
"windowsSoftwareIds": None
"productClassIds": None
"type": "LocalbootProduct"
"licenseRequired": False
"setupScript": "javavm.ins"
"updateScript": ""
"uninstallScript": "deljvm.ins"
"alwaysScript": ""
"onceScript": ""
"customScript": ""
```

Zu jedem Objekt gibt es eine Reihe von Operationen. In der Regel sind dies:

- *getObjects* (liefert die Objekte)
- *getHashes* (Variante, die aus Performance-Gründen die Backend-Objekte ohne Erzeugung von internen Objekten readonly durchreicht; bei entsprechenden Datenvolumen erheblich schneller als *getObjects*)
- *create* (zum komfortablen Erzeugen eines Objektes)
- *createObjects* (zum Erzeugen vieler Objekte)
- *delete* (zum Löschen eines Objektes)
- *deleteObjects* (zum Löschen vieler Objekte)
- *getIdents* (liefert nur die Objekt-Ids)
- *insertObject* (zum Erzeugen eines neuen Objektes)
- *updateObject* (zum Aktualisieren eines Objektes, erzeugt das Objekt, wenn nicht vorhanden)
- *updateObjects* (zum Aktualisieren vieler Objekte)

Die Namen der Methoden setzen sich zusammen aus:

<object name>_<operation>

Dadurch unterscheiden sie sich von den *Legacy* Methoden aus opsi 3.x welche in der Regel mit *get*, *set* oder *create* anfangen.

Die *getObjects*-Methoden haben zwei optionale Parameter:

- *attributes*
- *filter*

attributes dient dazu, nur bestimmte Attribute des Objektes abzufragen. Zurückgeliefert werden immer alle Attributname, aber nur die Werte der Attribute, welche das Objekt eindeutig kennzeichnen sowie die in *attributes* angegebenen Attributwerden. Die restlichen Attribute werden mit dem Wert *None* geliefert.

So liefert z.B die Methode *product_getObjects*, parametrisiert mit *attributes:["name"]* für das Produkt *javavm*:

```
"onceScript": None,
"ident": "javavm;1.6.0.20;2",
"windowsSoftwareIds": None,
"description": None,
"setupScript": None,
"changelog": None,
"customScript": None,
"advice": None,
"uninstallScript": None,
"userLoginScript": None,
"name": "Sun Java Runtime Environment",
"priority": None,
"packageVersion": "2",
"productVersion": "1.6.0.20",
"updateScript": None,
"productClassIds": None,
"alwaysScript": None,
"type": "LocalbootProduct",
"id": "javavm",
"licenseRequired": None
```

Wenn Sie keine *attributes*, aber einen *filter* angeben möchten, darf das Feld Sie für *attributes* nicht ganz leer bleiben, sondern muss den Wert *[]* erhalten.

Mit *filter* kann eingeschränkt werden, zu welchen Objekten Informationen geholt werden sollen, ähnlich einer sql-where-Bedingung.So schränkt für *product_getObjects* der Filter *{ "id":"javavm" }* die Rückgabe auf das Object *javavm* ein.

Bei den Methoden, denen ein oder mehrere Objekte übergeben werden, muss dies als JSON-Objekt bzw. als Liste von JSON-Objekten geschehen.

Die wichtigsten Objekte sind:

- *auditHardwareOnHost* (clientspezifische Hardwareinformationen)
- *auditHardware* (clientunabhängige Hardwareinformationen)
- *auditSoftwareOnClient* (clientspezifische Softwareinformationen)
- *auditSoftware* (clientunabhängige Softwareinformationen)
- *auditSoftwareToLicensePool* (Lizenzmanagement)
- *configState* (Verwaltung von Zusatzkonfigurationen)
- *config* (Verwaltung von neuen typisierten Zusatzkonfigurationen)
- *group* (Gruppenverwaltung)
- *host* (Server und Clients)
- *licenseContract* (Lizenzmanagement)
- *licenseOnClient* (Lizenzmanagement)
- *licensePool* (Lizenzmanagement)

- *objectToGroup* (Gruppenverwaltung)
- *productDependency* (Produktabhängigkeiten)
- *productOnClient* (Infos zu einem Produkt bezogen auf einen Client)
- *productOnDepot* (Infos zu einem Produkt bezogen auf ein Depot)
- *productPropertyState* (Depot und Client bezogene Product Property Werte)
- *productProperty* (Definition der Product Properties)
- *product* (Produkt Metadaten)
- *softwareLicenseToLicensePool* (Lizenzmanagement)
- *softwareLicense* (Lizenzmanagement)

Daneben gibt es noch eine Reihe von weiteren Objekten mit speziellen Operationen. Das aufgeführte Design ermöglicht es:

- schnell Informationen zu einer großen Zahl von Objekten zu übertragen,
- dabei mit einheitlicher Syntax Daten zu filtern,
- die Informationen auf syntaktische Korrektheit der erzeugten Objekte zu prüfen.

Hierdurch wird eine verbesserte Stabilität und höhere Performanz erreicht.

5.1.2 Die Objekte zur Datenspeicherung

host (server und clients)

Beispiel für einen OpsiClient:

```
 method host_getObjects [] {"id":"xpclient.vmnat.local"}
[
          {
          "ident" : "xpclient.vmnat.local",
          "description" : "",
          "created" : "2012-03-22 12:13:52",
          "inventoryNumber" : "",
          "ipAddress" : "172.16.166.101",
          "notes" : "Created by opsi-deploy-client-agent at Wed, 24 Aug 2011 10:24:36",
          "oneTimePassword" : "",
          "lastSeen" : "2012-03-30 16:20:04",
          "hardwareAddress" : "00:0c:29:35:70:a7",
          "opsiHostKey" : "1234567890abcef1234567890abcdef",
          "type" : "OpsiClient",
          "id" : "xpclient.vmnat.local"
          }
]
```

Die meisten dieser Daten finden sich im *clients* tab des opsi-configed.

Mögliche Werte für `type`:

- *OpsiClient*
- *OpsiConfigserver* (was bedeutet, dies ist auch ein *OpsiDepotserver*)
- *OpsiDepotserver*

Der Server type hat andere und mehr Daten als ein Client..

Beispiel für einen server:

```
 method host_getObjects [] {"id":"sepiolina.vmnat.local"}
[
          {
          "masterDepotId" : null,
          "ident" : "sepiolina.vmnat.local",
          "networkAddress" : "172.16.166.0/255.255.255.128",
          "description" : "",
          "inventoryNumber" : "",
          "ipAddress" : "172.16.166.1",
          "repositoryRemoteUrl" : "webdavs://sepiolina.vmnat.local:4447/repository",
          "depotLocalUrl" : "file:///var/lib/opsi/depot",
          "isMasterDepot" : true,
          "notes" : "",
          "hardwareAddress" : null,
          "maxBandwidth" : 0,
          "repositoryLocalUrl" : "file:///var/lib/opsi/repository",
          "opsiHostKey" : "1234567890abcef1234567890abcdef",
          "type" : "OpsiConfigserver",
          "id" : "sepiolina.vmnat.local",
          "depotWebdavUrl" : "webdavs://sepiolina:4447/depot",
          "depotRemoteUrl" : "smb://sepiolina/opsi_depot"
          }
]
```

Die meisten dieser Daten finden sich in der *depot configuration* des opsi-configed.

group (Gruppen Verwaltung)

Beschreibt Gruppen und Ihre hierarchiche Struktur

Beispiel für ein `group` Objekt:

```
 method group_getObjects
 [
       {
          "ident" : "sub2",
          "description" : "sub2",
          "notes" : "",
          "parentGroupId" : null,
          "type" : "HostGroup",
          "id" : "sub2"
          },
          {
          "ident" : "subsub",
          "description" : "subsub",
          "notes" : "",
          "parentGroupId" : "sub2",
          "type" : "HostGroup",
          "id" : "subsub"
          }
]
```

objectToGroup (Gruppen Verwaltung)

Beschreibt die Mitgliedschaft von Obketen in Gruppen.

Es gibt *Hostgroups* und *Productgroups*

Beispiel für ein `objectToGroup` Objekt:

```
 method objectToGroup_getObjects
[
         {
          "groupType" : "HostGroup",
          "ident" : "HostGroup;sub2;win7.vmnat.local",
          "type" : "ObjectToGroup",
          "groupId" : "sub2",
          "objectId" : "win7.vmnat.local"
          },
          {
          "groupType" : "HostGroup",
          "ident" : "HostGroup;subsub;win7x64.vmnat.local",
          "type" : "ObjectToGroup",
          "groupId" : "subsub",
          "objectId" : "win7x64.vmnat.local"
          },
        {
          "groupType" : "ProductGroup",
          "ident" : "ProductGroup;opsiessentials;opsi-client-agent",
          "type" : "ObjectToGroup",
          "groupId" : "opsiessentials",
          "objectId" : "opsi-client-agent"
          },
          {
          "groupType" : "ProductGroup",
          "ident" : "ProductGroup;opsiessentials;opsi-winst",
          "type" : "ObjectToGroup",
          "groupId" : "opsiessentials",
          "objectId" : "opsi-winst"
          }
]
```

product (product meta data)

Beschreibt die Meta-Daten eines Produktes wie sie bei der Erstellung des Produktes definiert wurden.

Beispiel für ein `product` Objekt:

```
 method product_getObjects [] {"id":"jedit","productVersion":"4.5"}
[
          {
          "onceScript" : "",
          "ident" : "jedit;4.5;3",
          "windowsSoftwareIds" :
                    [

                    ],
          "description" : "jEdit with opsi-winst Syntax-Highlighting",
          "setupScript" : "setup.ins",
          "changelog" : "",
          "customScript" : "",
          "advice" : "",
          "uninstallScript" : "uninstall.ins",
          "userLoginScript" : "",
          "name" : "jEdit programmer's text editor",
          "priority" : 0,
          "packageVersion" : "3",
          "productVersion" : "4.5",
          "updateScript" : "update.ins",
          "productClassIds" :
                    [

                    ],
          "alwaysScript" : "",
          "type" : "LocalbootProduct",
          "id" : "jedit",
```

```
          "licenseRequired" : false
          }
]
```

Anmerkung
Im Fall von mehreren Depotservern, können hier unterschiedliche Versionen eines `product` auftauchen.

Die Eintragungen für *productClassIds* und *windowsSoftwareIds* werden im Moment nicht verwendet.

productProperty (Definition der product properties)

Beschreibt die properties eines `product` wie sie bei der Erstellung des Produktes definiert wurden.

Beispiel für ein `productProperty` Objekt:

```
 method productProperty_getObjects [] {"productId":"jedit","productVersion":"4.5"}
[
          {
          "ident" : "jedit;4.5;3;start_server",
          "description" : "Should the jedit derver started at every startup ?",
          "editable" : false,
          "defaultValues" :
                    [
                    false
                    ],
          "multiValue" : false,
          "productVersion" : "4.5",
          "possibleValues" :
                    [
                    false,
                    true
                    ],
          "packageVersion" : "3",
          "type" : "BoolProductProperty",
          "propertyId" : "start_server",
          "productId" : "jedit"
          }
]
```

Anmerkung
Die für einen Client verwendeten default Werte finden sich nicht hier, sondern werden Depotspezifisch in `productPropertyState` Objekten gespeichert.

productPropertyState (Depot oder Client spezifische product property settings)

Beschreibt:
* die default Werte eines product property auf einem Depot * die Client spezifischen settings eines product properies.

Beispiel für ein `productPropertyState` Objekt:

```
 method productPropertyState_getObjects [] {"productId":"jedit"}
[
          {
          "ident" : "jedit;start_server;sepiolina.vmnat.local",
          "objectId" : "sepiolina.vmnat.local",
          "values" :
                    [
                    false
```

```
                    ],
          "type" : "ProductPropertyState",
          "propertyId" : "start_server",
          "productId" : "jedit"
          },
         {
          "ident" : "jedit;start_server;xpclient.vmnat.local",
          "objectId" : "xpclient.vmnat.local",
          "values" :
                    [
                    true
                    ],
          "type" : "ProductPropertyState",
          "propertyId" : "start_server",
          "productId" : "jedit"
          }

]
```

productDependency (product Abhängigkeiten)

Beschreibt die Abhäningkeit eines Produktes zu einem anderen Produkt wie sie bei der Erstellung des Produktes definiert wurden.

Beispiel für ein `productDependency` Objekt:

```
method productDependency_getObjects [] {"productId":"jedit","productVersion":"4.5"}
[
          {
          "ident" : "jedit;4.5;3;setup;javavm",
          "productAction" : "setup",
          "requiredPackageVersion" : null,
          "requirementType" : "before",
          "requiredInstallationStatus" : "installed",
          "productVersion" : "4.5",
          "requiredProductId" : "javavm",
          "requiredAction" : null,
          "requiredProductVersion" : null,
          "type" : "ProductDependency",
          "packageVersion" : "3",
          "productId" : "jedit"
          }
]
```

productOnClient (client spezifische Informationen zu einem Produkt z.B. Installationsstatus)

Beschreibt welche Produkte in welchen Versionen auf welchem Client installiert sind.

Beispiel für ein `productOnClient` Objekt:

```
 method productOnClient_getObjects [] {"productId":"jedit","clientId":"xpclient.vmnat.local"}
[
          {
          "ident" : "jedit;LocalbootProduct;xpclient.vmnat.local",
          "actionProgress" : "",
          "actionResult" : "successful",
          "clientId" : "xpclient.vmnat.local",
          "modificationTime" : "2012-03-30 15:49:04",
          "actionRequest" : "none",
          "targetConfiguration" : "installed",
          "productVersion" : "4.5",
          "productType" : "LocalbootProduct",
          "lastAction" : "setup",
          "packageVersion" : "3",
```

```
          "actionSequence" : -1,
          "type" : "ProductOnClient",
          "installationStatus" : "installed",
          "productId" : "jedit"
          }
]
```

productOnDepot (depot spezifische Informationen zu einem Produkt)

Beschreibt welches Produkt in welcher Version auf welchem Depot installiert ist.

Beispiel für ein `productOnDepot` Objekt:

```
 method productOnDepot_getObjects [] {"productId":"jedit"}
[
          {
          "ident" : "jedit;LocalbootProduct;4.4.1;2;depotserver.vmnat.local",
          "locked" : false,
          "productVersion" : "4.4.1",
          "productType" : "LocalbootProduct",
          "depotId" : "depotserver.vmnat.local",
          "type" : "ProductOnDepot",
          "packageVersion" : "2",
          "productId" : "jedit"
          },
          {
          "ident" : "jedit;LocalbootProduct;4.5;3;sepiolina.vmnat.local",
          "locked" : false,
          "productVersion" : "4.5",
          "productType" : "LocalbootProduct",
          "depotId" : "sepiolina.vmnat.local",
          "type" : "ProductOnDepot",
          "packageVersion" : "3",
          "productId" : "jedit"
          }
]
```

Anmerkung
Im Fall von mehreren Depotservern, können hier unterschiedliche Versionen eines Produktes auftauchen.

config (Verwaltung der Defaultwerte der Hostparameter)

Beschreibt die *Hostparameter* der *Server Konfiguration* des opsi-configeds.

Beispiel für ein `config` Objekt:

```
 method config_getObjects [] {"id":"opsiclientd.event_gui_startup.active"}
[
          {
          "ident" : "opsiclientd.event_gui_startup.active",
          "description" : "gui_startup active",
          "defaultValues" :
                    [
                    true
                    ],
          "editable" : false,
          "multiValue" : false,
          "possibleValues" :
                    [
                    false,
                    true
                    ],
```

```
          "type" : "BoolConfig",
          "id" : "opsiclientd.event_gui_startup.active"
          }
]
```

configState (Verwaltung der clientspezifischen Hostparameter)

Beschreibt die *Hostparameter* der *Client Konfiguration* des opsi-configeds..

Beispiel für ein `configState` Objekt:

```
 method configState_getObjects [] {"configId":"opsiclientd.event_gui_startup.active"}
[
          {
          "configId" : "opsiclientd.event_gui_startup.active",
          "ident" : "opsiclientd.event_gui_startup.active;wanclient.vmnat.local",
          "values" :
                    [
                    false
                    ],
          "objectId" : "wanclient.vmnat.local",
          "type" : "ConfigState"
          }
]
```

Anmerkung
Ein *configState* Objekt kann nicht erzeugt werden ohne das das *config* Objekt existiert auf das es referenziert.

auditHardwareOnHost (Clientspezifische Hardware Informationen)

Beschreibt die ermittelten Hardwaretypen (inclusive der clientspezifischen Daten). Die Idee ist in diesem Objekt die clientspezifischen Daten zu halten und in `auditHardware` nur die allgemeinen, so dass es dort z.B. nur einen Eintrag für eine Netzwerkkarte gibt, die in vielen Clients benutzt wird.
Leider funktioniert diese Idee in der Praxis nicht wirklich.

Beispiel für ein `auditHardwareOnHost` Objekt:

```
 method auditHardwareOnHost_getObjects [] {"hostId":"xpclient.vmnat.local","hardwareClass":"NETWORK_CONTROLLER","\
    ipAddress":"172.16.166.101"}
[
          {
          "vendorId" : "1022",
          "macAddress" : "00:0C:29:35:70:A7",
          "hardwareClass" : "NETWORK_CONTROLLER",
          "state" : 1,
          "deviceType" : "PCI",
          "subsystemVendorId" : "2000",
          "ipEnabled" : "True",
          "type" : "AuditHardwareOnHost",
          "firstseen" : "2012-03-30 15:48:15",
          "revision" : "10",
          "hostId" : "xpclient.vmnat.local",
          "vendor" : "Advanced Micro Devices (AMD)",
          "description" : "Ethernetadapter der AMD-PCNET-Familie",
          "subsystemDeviceId" : "1022",
          "deviceId" : "2000",
          "autoSense" : null,
          "netConnectionStatus" : "Connected",
          "maxSpeed" : null,
          "name" : "Ethernetadapter der AMD-PCNET-Familie",
          "serialNumber" : null,
```

```
          "lastseen" : "2012-03-30 15:48:15",
          "model" : null,
          "ipAddress" : "172.16.166.101",
          "adapterType" : "Ethernet 802.3"
          },
          {
          "vendorId" : "1022",
          "macAddress" : "00:0C:29:35:70:A7",
          "hardwareClass" : "NETWORK_CONTROLLER",
          "state" : 0,
          "deviceType" : "PCI",
          "subsystemVendorId" : "2000",
          "ipEnabled" : "True",
          "type" : "AuditHardwareOnHost",
          "firstseen" : "2012-03-08 14:26:14",
          "revision" : "10",
          "hostId" : "xpclient.vmnat.local",
          "vendor" : "VMware, Inc.",
          "description" : "VMware Accelerated AMD PCNet Adapter",
          "subsystemDeviceId" : "1022",
          "deviceId" : "2000",
          "autoSense" : null,
          "netConnectionStatus" : "Connected",
          "maxSpeed" : null,
          "name" : "VMware Accelerated AMD PCNet Adapter",
          "serialNumber" : null,
          "lastseen" : "2012-03-10 14:47:15",
          "model" : null,
          "ipAddress" : "172.16.166.101",
          "adapterType" : "Ethernet 802.3"
          },
  {
          "vendorId" : "1022",
          "macAddress" : "00:0c:29:35:70:a7",
          "hardwareClass" : "NETWORK_CONTROLLER",
          "state" : 0,
          "deviceType" : null,
          "subsystemVendorId" : "1022",
          "ipEnabled" : null,
          "type" : "AuditHardwareOnHost",
          "firstseen" : "2012-02-29 15:43:21",
          "revision" : "10",
          "hostId" : "xpclient.vmnat.local",
          "vendor" : "Advanced Micro Devices [AMD]",
          "description" : "Ethernet interface",
          "subsystemDeviceId" : "2000",
          "deviceId" : "2000",
          "autoSense" : "",
          "netConnectionStatus" : "yes",
          "maxSpeed" : null,
          "name" : "79c970 [PCnet32 LANCE]",
          "serialNumber" : "00:0c:29:35:70:a7",
          "lastseen" : "2012-03-30 14:58:30",
          "model" : "79c970 [PCnet32 LANCE]",
          "ipAddress" : "172.16.166.101",
          "adapterType" : ""
          }
]
```


auditHardware (Client unabhängige Hardware Informationen)

Beschreibt die ermittelten Hardwaretypen (ohne die clientspezifischen Daten). Die Idee ist in diesem Objekt nur die allgemeinen Daten eines Hardwaretyps zu halten, so dass es hier z.B. nur einen Eintrag für eine Netzwerkkarte gibt, die in vielen Clients benutzt wird.
Leider funktioniert diese Idee in der Praxis nicht wirklich.

Beispiel für ein `auditHardware` Objekt:

```
 method auditHardware_getObjects [] {"hardwareClass":"NETWORK_CONTROLLER","vendorId":"1022"}
[
          {
          "vendorId" : "1022",
          "deviceId" : "2000",
          "maxSpeed" : null,
          "vendor" : "Advanced Micro Devices [AMD]",
          "name" : "79c970 [PCnet32 LANCE]",
          "subsystemDeviceId" : "2000",
          "deviceType" : null,
          "subsystemVendorId" : "1022",
          "autoSense" : "",
          "model" : "79c970 [PCnet32 LANCE]",
          "revision" : "10",
          "type" : "AuditHardware",
          "hardwareClass" : "NETWORK_CONTROLLER",
          "adapterType" : "",
          "description" : "Ethernet interface"
          },
          {
          "vendorId" : "1022",
          "deviceId" : "2000",
          "maxSpeed" : null,
          "vendor" : "VMware, Inc.",
          "name" : "VMware Accelerated AMD PCNet Adapter",
          "subsystemDeviceId" : "1022",
          "deviceType" : "PCI",
          "subsystemVendorId" : "2000",
          "autoSense" : null,
          "model" : null,
          "revision" : "10",
          "type" : "AuditHardware",
          "hardwareClass" : "NETWORK_CONTROLLER",
          "adapterType" : "Ethernet 802.3",
          "description" : "VMware Accelerated AMD PCNet Adapter"
          },
          {
          "vendorId" : "1022",
          "deviceId" : "2000",
          "maxSpeed" : null,
          "vendor" : "Advanced Micro Devices (AMD)",
          "name" : "Ethernetadapter der AMD-PCNET-Familie",
          "subsystemDeviceId" : "1022",
          "deviceType" : "PCI",
          "subsystemVendorId" : "2000",
          "autoSense" : null,
          "model" : null,
          "revision" : "10",
          "type" : "AuditHardware",
          "hardwareClass" : "NETWORK_CONTROLLER",
          "adapterType" : "Ethernet 802.3",
          "description" : "Ethernetadapter der AMD-PCNET-Familie"
          },
 {
          "vendorId" : "1022",
          "deviceId" : "2000",
          "maxSpeed" : null,
          "vendor" : "Advanced Micro Devices (AMD)",
          "name" : "Ethernetadapter der AMD-PCNET-Familie",
          "subsystemDeviceId" : "1022",
          "deviceType" : "PCI",
          "subsystemVendorId" : "2000",
          "autoSense" : null,
          "model" : null,
          "revision" : "10",
          "type" : "AuditHardware",
```

```
          "hardwareClass" : "NETWORK_CONTROLLER",
          "adapterType" : "Ethernet 802.3",
          "description" : "Ethernetadapter der AMD-PCNET-Familie"
          },
          {
          "vendorId" : "1022",
          "deviceId" : "2000",
          "maxSpeed" : null,
          "vendor" : "Advanced Micro Devices (AMD)",
          "name" : null,
          "subsystemDeviceId" : "2000",
          "deviceType" : "PCI",
          "subsystemVendorId" : "1022",
          "autoSense" : null,
          "model" : "",
          "revision" : null,
          "type" : "AuditHardware",
          "hardwareClass" : "NETWORK_CONTROLLER",
          "adapterType" : null,
          "description" : "Ethernetadapter der AMD-PCNET-Familie"
          },
(....)
[
```

auditSoftwareOnClient (Clientspezifische Software Informationen)

Beschreibt die ermittelten Softwaretypen (inclusive der clientspezifischen Daten). Die Idee ist in diesem Objekt die clientspezifischen Daten zu halten und in `auditSoftware` nur die allgemeinen, so dass es dort z.B. nur einen Eintrag für eine Office-Software gibt, die in vielen Clients benutzt wird.

Beispiel für ein `auditSoftwareOnClient` Objekt:

```
 method auditSoftwareOnClient_getObjects  [] {"name":"jEdit 4.5.0","clientId":"xpclient.vmnat.local"}
[
          {
          "ident" : "jEdit 4.5.0;4.5.0;;;x86;xpclient.vmnat.local",
          "licenseKey" : "",
          "name" : "jEdit 4.5.0",
          "uninstallString" : "\\\"C:\\\\Programme\\\\jEdit\\\\unins000.exe\\\"",
          "usageFrequency" : -1,
          "clientId" : "xpclient.vmnat.local",
          "lastUsed" : "0000-00-00 00:00:00",
          "subVersion" : "",
          "language" : "",
          "state" : 1,
          "version" : "4.5.0",
          "lastseen" : "2012-03-30 16:19:55",
          "binaryName" : "",
          "type" : "AuditSoftwareOnClient",
          "firstseen" : "2012-03-30 16:19:55",
          "architecture" : "x86"
          }
]
```

auditSoftware (Client unahängige Software Informationen)

Beschreibt die ermittelten Softwaretypen (ohne die clientspezifischen Daten). Die Idee ist in diesem Objekt nur die allgemeinen Daten eines Softwaretyps zu halten, so dass es hier z.B. nur einen Eintrag für eine Office-Software gibt, die in vielen Clients benutzt wird.

Beispiel für ein `auditSoftware` Objekt:

```
 method auditSoftware_getObjects  [] {"name":"jEdit 4.5.0"}
[
```

```
          {
          "windowsDisplayVersion" : "4.5.0",
          "ident" : "jEdit 4.5.0;4.5.0;;;x64",
          "name" : "jEdit 4.5.0",
          "windowsSoftwareId" : "jedit_is1",
          "windowsDisplayName" : "jEdit 4.5.0",
          "installSize" : -1,
          "subVersion" : "",
          "language" : "",
          "version" : "4.5.0",
          "architecture" : "x64",
          "type" : "AuditSoftware"
          },
          {
          "windowsDisplayVersion" : "4.5.0",
          "ident" : "jEdit 4.5.0;4.5.0;;;x86",
          "name" : "jEdit 4.5.0",
          "windowsSoftwareId" : "jedit_is1",
          "windowsDisplayName" : "jEdit 4.5.0",
          "installSize" : -1,
          "subVersion" : "",
          "language" : "",
          "version" : "4.5.0",
          "architecture" : "x86",
          "type" : "AuditSoftware"
          }
]
```


auditSoftwareToLicensePool (Lizenzmanagement)

Beschreibt die Zuordnung von Mustern aus der Softwareinventarisierung (*auditSoftware*) zu einzelnen Lizenzpools.

Beispiel für ein `auditSoftwareToLicensePool` Objekt:


```
 method auditSoftwareToLicensePool_getObjects [] {"licensePoolId":"win7-msdn-prof"}
[
          {
          "ident" : "Windows 7 Professional N;6.1;00376-165;de-DE;x64;win7-msdn-prof",
          "name" : "Windows 7 Professional N",
          "language" : "de-DE",
          "subVersion" : "00376-165",
          "licensePoolId" : "win7-msdn-prof",
          "version" : "6.1",
          "architecture" : "x64",
          "type" : "AuditSoftwareToLicensePool"
          },
          {
          "ident" : "Windows 7 Professional N;6.1;00376-165;de-DE;x86;win7-msdn-prof",
          "name" : "Windows 7 Professional N",
          "language" : "de-DE",
          "subVersion" : "00376-165",
          "licensePoolId" : "win7-msdn-prof",
          "version" : "6.1",
          "architecture" : "x86",
          "type" : "AuditSoftwareToLicensePool"
          }
]
```


softwareLicenseToLicensePool (Lizenzmanagement)

Beschreibt die Zuordnung von 'softwareLicenseId's zu 'licensePoolId's.

Beispiel für ein `softwareLicenseToLicensePool` Objekt:

```
method softwareLicenseToLicensePool_getObjects [] {"licensePoolId":"win7-msdn-prof"}
[
          {
          "licensePoolId" : "win7-msdn-prof",
          "softwareLicenseId" : "uib-msdn-win7-vol",
          "ident" : "uib-msdn-win7-vol;win7-msdn-prof",
          "licenseKey" : "12345-12345-12345-12345-3dbv6",
          "type" : "SoftwareLicenseToLicensePool"
          }
]
```

softwareLicense (Lizenzmanagement)

Beschreibt die existierenden Softwarelizenzen und deren Metadaten.

Beispiel für ein `softwareLicense` Objekt:

```
 method softwareLicense_getObjects [] {"id":"uib-msdn-win7-vol"}
[
          {
          "ident" : "uib-msdn-win7-vol;msdn-uib",
          "maxInstallations" : 0,
          "boundToHost" : null,
          "expirationDate" : "0000-00-00 00:00:00",
          "licenseContractId" : "msdn-uib",
          "type" : "VolumeSoftwareLicense",
          "id" : "uib-msdn-win7-vol"
          }
]
```

licenseContract (Lizenzmanagement)

Beschreibt die existierenden Lizenzverträge und deren Metadaten.

Beispiel für ein `licenseContract` Objekt:

```
 method licenseContract_getObjects [] {"id":"msdn-uib"}
[
          {
          "ident" : "msdn-uib",
          "description" : "",
          "conclusionDate" : "2011-04-22 00:00:00",
          "notificationDate" : "0000-00-00 00:00:00",
          "notes" : "",
          "expirationDate" : "0000-00-00 00:00:00",
          "partner" : "Microsoft",
          "type" : "LicenseContract",
          "id" : "msdn-uib"
          }
]
```

licenseOnClient (Lizenzmanagement)

Beschreibt welcher Client welche Lizenz in Verwendung hat.

Beispiel für ein `licenseOnClient` Objekt:

```
 method licenseOnClient_getObjects  [] {"clientId":"win7client.vmnat.local"}
[
          {
          "softwareLicenseId" : "uib-msdn-win7-vol",
          "ident" : "uib-msdn-win7-vol;win7-msdn-prof;win7client.vmnat.local",
```

```
          "licenseKey" : "12345-12345-12345-12345-3dbv6",
          "notes" : "",
          "clientId" : "win7client.vmnat.local",
          "licensePoolId" : "win7-msdn-prof",
          "type" : "LicenseOnClient"
          }
]
```

licensePool (Lizenzmanagement)

Beschreibt einen Lizenzpool und dessen Zuordnung zu Produkten.

Beispiel für ein `licensePool` Objekt:

```
 method licensePool_getObjects [] {"id":"win7-msdn-prof"}
[
          {
          "ident" : "win7-msdn-prof",
          "type" : "LicensePool",
          "description" : "MSDN Keys",
          "productIds" :
                    [
                    "win7",
                    "win7-x64"
                    ],
          "id" : "win7-msdn-prof"
          }
]
```

5.1.3 Die spezial Objekte

Anmerkung
This chapter has to be written ...

5.2 opsi3-Methoden

Die mit opsi 3 eingeführten Methoden stehen als *Legacy* Methoden weiterhin zur Verfügung, werden aber ab opsi 4.0 intern auf die neuen Methoden *gemappt*.

Hier eine Liste der Methoden (dargestellt in der Form des Aufrufs mit *opsi-admin*) mit einer kurzen Beschreibung. Diese dient zur Orientierung und nicht als Referenz. Das bedeutet die Beschreibung muss nicht alle Informationen enthalten, die Sie benötigen, um diese Methode tatsächlich zu verwenden.

```
method addHardwareInformation hostId, info
```

Fügt Hardwareinformationen zum Rechner *hostid* hinzu. Übergeben wird der Hash *info*. Vorhandene Informationen werden überschrieben, wenn die Keys über einstimmen. Es sind nur bestimmte Keys zulässig

```
method authenticated
```

Überprüfen ob die Authentifizierung am Service erfolgreich war.

```
method checkForErrors
```

Überprüft auf Inkonsistenzen im Backend (bisher nur implementiert für Backend File)

```
method createClient clientName, domain, description=None, notes=None
```

Erzeugt einen neuen Client.

```
method createGroup groupId, members = [], description = ""
```

Erzeugt eine Gruppe von Clients wie sie vom opsi-configed verwendet wird.

```
method createLicenseKey productId, licenseKey
```

Weist dem Produkt *produktid* einen (weiteren) Lizenzkey zu.

```
method createLocalBootProduct productId, name, productVersion, packageVersion, licenseRequired=0, setupScript="", \
    uninstallScript="", updateScript="", alwaysScript="", onceScript="", priority=10, description="", advice="", \
    productClassNames=('localBoot')
```

Legt ein neues Localboot-Produkt (Winst-Produkt) an.

```
method createNetBootProduct productId, name, productVersion, packageVersion, licenseRequired=0, setupScript="", \
    uninstallScript="", updateScript="", alwaysScript="", onceScript="", priority=10, description="", advice="", \
    productClassNames=('netboot')
```

Legt ein neues bootimage Produkt an

```
method createOpsiBase
```

Nur für interne Verwendung beim LDAP-Backend.

```
method createProduct productType, productId, name, productVersion, packageVersion, licenseRequired=0,setupScript="", \
    uninstallScript="", updateScript="", alwaysScript="", onceScript="", priority=10, description="", advice="", \
    productClassNames=""
```

Legt ein neues Produkt an.

```
method createProductDependency productId, action, requiredProductId="", requiredProductClassId="", requiredAction="", \
    requiredInstallationStatus="", requirementType=""
```

Erstellt Produktabhängigkeiten.

```
method createProductPropertyDefinition productId, name, description=None, defaultValue=None, possibleValues=[]
```

Erstellt eine Produkteigenschaft.

```
method createServer serverName, domain, description=None
```

Erstellt im LDAP-Backend einen neuen Server.

```
method createServerProduct  productId, name, productVersion, packageVersion, licenseRequired=0,setupScript="", \
    uninstallScript="", updateScript="", alwaysScript="", onceScript="", priority=10, description="", advice="", \
    productClassNames=('server')
```

Noch nicht implementiert - für zukünftige Verwendung.

```
method deleteClient clientId
```

Löscht einen Client.

```
method deleteGeneralConfig objectId
```

Löscht Konfiguration eines Clients oder einer Domain.

```
method deleteGroup groupId
```

Löscht eine Clientgruppe.

```
method deleteHardwareInformation hostId
```

Löscht sämtliche Hardwareinfos zum Rechner *hostid*.

```
method deleteLicenseKey productId, licenseKey
```

Löscht einen Lizenzkey.

```
method deleteNetworkConfig objectId
```

Löscht Netzwerkkonfiguration (z.B. depotshare Eintrag) für Client oder Domain.

```
method deleteOpsiHostKey hostId
```

Löscht einen pckey aus der pckey-Datenbank.

```
method deleteProduct productId
```

Löscht ein Produkt aus der Datenbasis.

```
method deleteProductDependency productId, action, requiredProductId="", requiredProductClassId="", requirementType=""
```

Löscht Produktabhängigkeit.

```
method deleteProductProperties productId *objectId
```

Löscht alle Properties eines Produkts.

```
method deleteProductProperty productId property *objectId
```

Löscht ein Property eines Produkts.

```
method deleteProductPropertyDefinition productId, name
method deleteProductPropertyDefinitions productId
```

Löscht alle Produkteigenschaften zum Produkt *productid*.

```
method deleteServer serverId
```

Löscht die Serverkonfiguration.

```
method exit
```

Verlässt den opsi-admin.

```
method getBackendInfos_listOfHashes
```

Liefert eine Beschreibung der auf dem opsi-server konfigurierten Backends und welche davon aktiviert sind.

```
method getBootimages_list
```

Liefert die Liste der zur Auswahl stehenden Bootimages.

```
method getClientIds_list serverId = None, groupId = None, productId = None, installationStatus = None, actionRequest = \
    None
```

Liefert die Liste der Clients, welche den angegebenen Kriterien entsprechen.

```
method getClients_listOfHashes serverId = None, groupId = None, productId = None, installationStatus = None, \
    actionRequest = No
```

Liefert die Liste der Clients, welche den angegebenen Kriterien entsprechen, zusammen mit Beschreibung, Notizen und *Lastseen*.

```
method getDefaultNetBootProductId clientId
```

Liefert das Netboot-Produkt (z.B. Betriebssystem) welches beim Aufruf des bootimages *install* installiert wird.

```
method getDomain hostId
```

Liefert die Domain zu einem Rechner.

```
method getGeneralConfig_hash objectId
```

Liefert allgemeine Konfiguration zu einem Client oder einer Domain.

```
method getGroupIds_list
```

Liefert die Liste der gespeicherten Clientgruppen.

```
opsi-admin -d -S method auditHardwareOnHost_getObjects '[]' '{"hostId":"<hostId>"}'
```

Liefert die Hardwareinformationen zu dem angegebenen Rechner.

```
method getHostId hostname
```

Liefert hostid zu dem angegebenen Hostnamen.

```
method getHost_hash hostId
```

Liste der Eigenschaften des angegebenen Rechners.

```
method getHostname hostId
```

Liefert hostname zur Host-ID.

```
method getInstallableLocalBootProductIds_list clientId
```

Liefert alle Localboot-Produkte, die auf diesem Client installiert werden können.

```
method getInstallableNetBootProductIds_list clientId
```

Liefert alle Netboot-Produkte, die auf diesem Client installiert werden können.

```
method getInstallableProductIds_list clientId
```

Liefert alle Produkte, die auf diesem Client installiert werden können.

```
method getInstalledLocalBootProductIds_list hostId
```

Liefert alle Localboot-Produkte, die auf diesem Client installiert sind.

```
method getInstalledNetBootProductIds_list hostId
```

Liefert die Liste der installierten Netboot-Produkte für einen Client oder Server.

```
method getInstalledProductIds_list hostId
```

Liefert die Liste der installierten Produkte für einen Client oder Server.

```
method getIpAddress hostId
```

Liefert IP-Adresse zur Host-ID.

```
method getLicenseKey productId, clientId
```

Liefert einen freien Lizenzkey zu dem angegebenen Produkt bzw. liefert den der *clientId* zugeordneten Lizenzkey,

```
method getLicenseKeys_listOfHashes productId
```

Liefert eine Liste der Lizenzkeys für das angegebene Produkt.

```
method getLocalBootProductIds_list
```

Liefert alle (z.B. im LDAP-Baum) bekannten Localboot-Produkte.

```
method getLocalBootProductStates_hash clientIds = []
```

Liefert für die angegebenen Clients Installationsstatus und Action-Requests für alle Localboot-Produkte.

```
method getMacAddresses_list hostId
```

Liefert die MAC-Adresse zum angegebenen Rechner.

```
method getNetBootProductIds_list
```

Liefert Liste der Netboot-Produkte.

```
method getNetBootProductStates_hash clientIds = []
```

Liefert für die angegebenen Clients Installationsstatus und {Action-request= für alle Netboot-Produkte.

```
method getNetworkConfig_hash objectId
```

Liefert die Netzwerk-spezifischen Konfigurationen für einen Client oder eine Domain.

```
method getOpsiHostKey hostId
```

Liefert den pckey zur angegeben Host-ID.

```
method getPcpatchPassword hostId
```

Liefert das mit dem *pckey* von *hostId* verschlüsselte Passwort des Users *pcpatch*.

```
method getPossibleMethods_listOfHashes
```

Liefert die Liste der aufrufbaren Methoden (in etwa so wie in diesem Kapitel beschrieben).

```
method getPossibleProductActionRequests_list
```

Liefert die Liste der in opsi prinzipiell zulässigen Action-Requests.

```
method getPossibleProductActions_hash
```

Liefert zu allen Produkten die möglichen Aktionen (*setup*, *deinstall*,...).

```
method getPossibleProductActions_list productId=softprod
```

Liefert zum angegebenen Produkt die möglichen Aktionen (*setup*, *deinstall*,...).

```
method getPossibleProductInstallationStatus_list
```

Liefert die möglichen Installationsstatus (*installed*, *not_installed*,...).

```
method getPossibleRequirementTypes_list
```

Liefert die möglichen Typen von Produktabhängigkeiten (*before*, *after*, ...).

```
method getProductActionRequests_listOfHashes clientId
```

Liefert die anstehenden ausführbaren Aktionen für den angegebenen Client.

```
method getProductDependencies_listOfHashes productId = None
```

Liefert die bekannten Produktabhängigkeiten (zum angegebenen Produkt).

```
method getProductIds_list productType = None, hostId = None, installationStatus = None
```

Liefert die Liste der Produkte, die den angegebenen Kriterien entsprechen.

```
method getProductInstallationStatus_hash productId, hostId
```

Liefert den Installationsstatus zum angegebenen Client und Produkt.

```
method getProductInstallationStatus_listOfHashes hostId
```

Liefert den Installationsstatus zum angegebenen Client.

```
method getProductProperties_hash productId, objectId = None
```

Liefert die Schalterstellungen (Product-Properties) zum angegebenen Produkt und Client.

```
method getProductPropertyDefinitions_hash
```

Liefert alle bekannten Product-Properties mit Beschreibung, erlaubten Werten etc..

```
method getProductPropertyDefinitions_listOfHashes productId
```

Liefert die Product-Properties zum angegebenen Produkt mit Beschreibung, erlaubten Werten etc..

```
method getProductStates_hash clientIds = []
```

Liefert Installationsstatus und Action-Requests der einzelnen Produkte (zu den agegebenen Clients).

```
method getProduct_hash productId
```

Liefert die Metadaten (Beschreibung, Version,...) zum angegebenen Produkt.

```
method getProvidedLocalBootProductIds_list serverId
```

Liefert die Liste der auf dem angegebenen Server bereitgestellten Localboot-Produkte.

```
method getProvidedNetBootProductIds_list serverId
```

Liefert die Liste der auf dem angegebenen Server bereitgestellten Netboot-Produkte.

```
method getServerId clientId
```

Liefert den zuständigen opsi-configserver zum angegebenen Client.

```
method getServerIds_list
```

Liefert die Liste der bekannten opsi-configserver.

```
method getServerProductIds_list
```

Liste der Server-Produkte.

```
method getUninstalledProductIds_list hostId
```

Liefert die deinstallierten Produkte.

```
method powerOnHost mac
```

Sendet ein WakeOnLan-Signal an die angegebene MAC.

```
method setBootimage bootimage, hostId, mac=None
```

Setzt *bootimage* als Bootimage beim Start des angegebenen Clients.

```
method setGeneralConfig config, objectId = None
```

Setzt für Client oder Domain die GenaralConfig.

```
method setHostDescription hostId, description
```

Setzt für einen Client die Beschreibung.

```
method setHostLastSeen hostId, timestamp
```

Setzt für einen Client den Zeitstempel für *LastSeen*.

```
method setHostNotes hostId, notes
```

Setzt für einen Client die Notiz-Angaben.

```
method setMacAddresses hostId, macs
```

Trägt für einen Client seine MAC-Adresse in die Datenbank ein.

```
method setNetworkConfig objectId, serverId='', configDrive='', configUrl='', depotDrive='', depotUrl='', utilsDrive='',\
      utilsUrl='', winDomain='', nextBootServiceURL=''
```

Setzt für einen Client die angegebene Netzwerkdaten für den opsi-client-agent.

```
method setOpsiHostKey hostId, opsiHostKey
```

Setzt für einen Rechner den *pckey*.

```
method setPXEBootConfiguration hostId *args
```

Schreibt die Pipe für dne PXE-Boot mit *args in der *append*-Liste.

```
method setPcpatchPassword hostId password
```

Setzt das verschlüsselte (!) *password* für *hostId*.

```
method setProductActionRequest productId, clientId, actionRequest
```

Setzt für den angegebenen Client und das angegebene Produkt einen Action-Request.

```
method setProductInstallationStatus productId, hostId, installationStatus, policyId="", licenseKey=""
```

Setzt für den angegebenen Client und das angegebene Produkt einen Installationsstatus.

```
method setProductProperties productId, properties, objectId = None
```

Setzt Product-Properties für das angegebene Produkt (und den angegebenen Client).

```
method unsetBootimage hostId
```

Setzt einen Bootimage-Start für den angegebenen Client zurück.

```
method unsetPXEBootConfiguration hostId
```

Löscht die PXE-Boot-Pipe.

```
method unsetProductActionRequest productId, clientId
```

Setzt einen Action-Request auf *none*.

5.3 Backend-Erweiterungen

Durch die in opsi 4 implementierten Funktionalität des Backend-Extenders können auf der Basis der opsi4-API-Methoden oder des Funktionskerns jederzeit zusätzliche Methoden definiert und als API-Erweiterung aufrufbar gemacht werden.

Das Standard-API-Methoden-Set wird durch den *opsiconfd* mittels Überlagerung der in den Python-Dateien in */etc/opsi/backendManager/extend.d* definierten Methoden erstellt.

Zusätzliche Methoden-Sets mit Aufruf per speziellem Pfad können zudem in Dateien in Unterverzeichnissen dieses Verzeichnisses definiert werden. Standardmäßig ist hier ein Set von Methoden, die speziell auf Datenanforderungen durch den *opsi-configed* zugeschnitten sind, im Verzeichnis */etc/opsi/backendManager/extend.d/configed* implementiert. Im Web-Interface werden die entsprechenden Methoden (zusammen mit den Standardmethoden) sichtbar, wenn als *Path* in der Drop-down-Liste *interface/extend/configed* ausgewählt wird.

Backend-Erweiterungen können auch dazu dienen für spezifische Konfigurationsaufgaben angepasste Zusatzfunktionen zu implementieren.

6 Freischaltung kostenpflichtiger Module

Auch wenn opsi Open Source ist, so gibt es einige Zusatzkomponenten, die im Rahmen eines Kofinanzierungsprojektes erstellt wurden und evtl. noch nicht Open Source bzw. noch nicht kostenlos sind.

Zur Zeit (Juni 2012) sind dies:

- MySQL-Backend für Konfigurationsdaten (siehe Abschnitt 21.3.2)
- opsi-Lizenzmanagement Modul (siehe Abschnitt 14)
- Unterstützung für Clients über WAN/VPN (siehe Abschnitt 15)
- Dynamische Depotzuweisung (siehe Abschnitt 17)
- User Profile Management (siehe Abschnitt 19)
- opsi Nagios Connector (siehe Abschnitt 20)

Zu diesem Thema siehe auch: http://uib.de/www/kofinanziert/index.html

Sobald die Entwicklungskosten eingenommen sind, werden auch diese Module Open Source bzw. kostenlos sein. Um bis dahin die Verwendung dieser Module den zahlenden Kunden und zu Evaluierungszwecken zu gestatten, gibt es die Freischaltdatei `/etc/opsi/modules` , welche durch eine elektronische Signatur vor unautorisierter Veränderung geschützt ist. Ist diese Datei nicht vorhanden, so funktionieren nur die *freien* Module von opsi.

Um zu Evaluierungszwecken eine zeitlich befristet gültige Freischaltdatei zu erhalten,wenden sich an info@uib.de. Im Rahmen einer Beteiligung an den entsprechenden Kofinanzierungsprojekten erhalten Sie eine unbefristet gültige Freischaltdatei.

Führen Sie danach den folgenden Befehl aus:

```
opsi-setup --set-rights /etc/opsi
```

Kontrollieren Sie die Freischaltung mit einer der folgenden Methoden:

Im opsi-configed können Sie über den Menüpunkt Hilfe/opsi-Module sich den Status Ihrer Freischaltung anzeigen lassen.

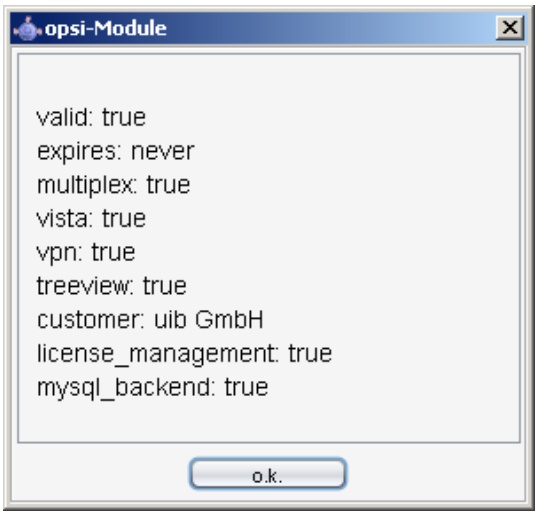


Abbildung 40: Anzeige der Freischaltung im opsi-configed

Mit der Methode `backend_info` können Sie mit opsi-admin überprüfen, welche Module freigeschaltet sind. (Hinweis: Geben Sie weder die Datei noch die Ausgabe dieses Befehls öffentlich weiter, zumindest nicht ohne die Signatur zu löschen).

```
opsi-admin -d method backend_info
{
"opsiVersion" : "3.99.0.0",
"modules" :
{
"customer" : "uib GmbH",
"vista" : true,
"vpn" : true,
"license_management" : true,
"expires" : "never",
"valid" : true,
"multiplex" : true,
"signature" : "DIES-IST-KEINE-ECHTE-SIGNATUR",
"treeview" : true,
"mysql_backend" : true
}
}
```

7 opsi-client-agent

7.1 Überblick

Damit die Verteilung von Software nicht zur "Turnschuh-Administration" wird, muss ein Client-PC selbstständig erkennen, dass neue Softwarepakete oder Updates für ihn bereit stehen und diese installieren. Bei der Installation ist auf jede Form von Anwender-Interaktion zu verzichten, damit diese unbeaufsichtigt erfolgen kann und nicht durch verunsicherte Anwender notwendige Installationen abgebrochen werden.

Diese Anforderungen werden bei opsi durch einen Agenten auf dem Client realisiert:

Auf dem Client wird der sogenannte *opsi-client-agent* installiert. Dieser überprüft üblicherweise beim Start des Clients und vor dem Login des Anwenders, anhand von Konfigurations-Informationen auf dem *opsi-configserver*, ob für diesen Client ein Update installiert werden soll.

Soll Software installiert werden, wird das skriptgesteuerte Installationsprogramm *opsi-winst* gestartet. Auf einer Dateifreigabe, dem sogenannten *opsi-depot*, stehen die dafür notwendigen Skripte und Softwarepakete bereit. Während dieser Zeit besteht für den Anwender keine Notwendigkeit und keine Möglichkeit in den Installationsprozess einzugreifen.

Um zu verhindern, dass sich ein Anwender vor dem Abschluss der Installation am System anmelden und so den Installations-Prozess stören kann, wird zusätzlich der sogenannte *opsi-Loginblocker* installiert, der eine Anmeldung erst nach Abschluss der Installationen zulässt.

Damit Softwarepakete mit dem Programm *opsi-winst* ohne Interaktion installiert werden können, müssen sie dafür vorbereitet werden. Siehe dazu das Kapitel *Einbindung eigener Software in die Softwareverteilung von opsi* im *opsi-getting-started* Handbuch.

7.2 Verzeichnisse des opsi-client-agent

Der *opsi-client-agent* installiert sich nach `%ProgramFiles%\opsi.org\opsi-client-agent`.

Dieses Verzeichnis enthält alle Programme des *opsi-client-agent* wie z.B. den *opsiclientd*, die *opsiclientd notifier* den *opsi-winst* und einige Bibliotheken. Weiterhin finden sich hier die Konfigurationsdateien und grafischen Vorlagen der genannten Programme.
Das Verzeichnis `%ProgramFiles%\opsi.org\opsi-client-agent` ist gegen Veränderung mit Benutzerrechten geschützt.
Das Verzeichnis `%ProgramFiles%\opsi.org\opsi-client-agent\opsiclientd` enthält die Konfigurationsdatei des *opsiclientd* und kann nur mit Administratorrechten gelesen werden.

Weiterhin gibt es das Verzeichnis `c:\opsi.org`.

Dieses Verzeichnis dient zur Zeit für den Installationscache (wenn gecached installiert wird → WAN-Erweiterung). Es wird in Zukunft noch weitere Funktionen übernehmen.
Das Verzeichnis `c:\opsi.org` kann nur mit Administratorrechten gelesen werden.

Logdateien des *opsi-client-agent* finden sich zur Zeit unter `c:\tmp`.

7.3 Der Service: opsiclientd

Der *opsiclientd* ist die Basis des *opsi-client-agents*. Er läuft als Service mit administrativen Rechten und wird beim Boot automatisch gestartet.

Wesentliche Funktionen sind:

- Eventbasierte Steuerung: Es kann auf unterschiedliche Events im System reagiert werden. Ein Event ist zum Beispiel der Start des Betriebssystems.
- Steuerung über Webservice: Auf den Webservice des *opsiclientd* kann über das Netzwerk zugegriffen werden. Diese Schnittstelle dient zum Anstoßen von Installationen (*push*) aber auch zu Wartungszwecken.
- Remote Konfiguration: Alle wesentlichen Konfigurationsdaten des *opsiclientd* lassen sich zentral über die *Host-Parameter* global oder Client-spezifisch bearbeiten.

Der *opsi-client-agent* besteht aus mehreren Komponenten:

- *opsiclientd*: Der zentrale Service des *opsi-client-agents*.
- *opsiclientd notifier*: Fenster zur Information / Kommunikation mit dem Anwender
- *opsi-Loginblocker*: Sperrt den Login bis die Installationen abgeschlossen sind.

7.3.1 Installation

Im Rahmen einer Neuinstallation eines Betriebssystems per unattended Setup über opsi wird der *opsi-client-agent* automatisch mit installiert.

Zur Deinstallation kann der *opsi-client-agent* auf *uninstall* gesetzt werden.

Zur nachträglichen Installation oder zu Reparaturzwecken siehe Kapitel Abschnitt 7.5.

7.3.2 opsiclientd

Kernkomponente des *opsi-client-agents* ist der Service *opsiclientd*. Dieser läuft beim Start des Betriebssystems an.

Er übernimmt folgende Aufgaben:

- Während das System bootet und der opsiclientd auf den Start der Windows GUI wartet, wird der *block_login_notifier* ausgeführt. Dieser zeigt standardmäßig ein Schloss in der oberen rechten Ecke des Bildschirms.
- Er baut beim Auftreten der konfigurierten Events Kontakt zum *opsi-configserver* auf. Konfigurationen und anstehende *Action-Requests* werden per JSON-RPC abgefragt. Das Standard-Event ist hierbei *gui_startup*, welches beim Start des Rechners (Start der GUI) und damit vor dem Login aktiv wird.
- Er stellt eine Named-Pipe bereit, über die der *opsi-Loginblocker* Kontakt zu ihm aufnehmen kann, um den Zeitpunkt der Freigabe des Logins zu erfragen. Auch diese Kommunikation erfolgt per *JSON-RPC*.
- Startet den *opsiclientd notifier* zur Interaktion und Kommunikation mit dem Anwender.
- Bei Bedarf stellt er eine Verbindung zum *opsi-depot* her, aktualisiert die lokale Installation des *opsi-winst* und startet diesen zur Bearbeitung der anstehenden *Action-Requests* (Installationen).

7.3.3 opsiclientd notifier

Der *opsiclientd notifier* realisiert die Interaktion mit dem Anwender. Hier werden sowohl Statusmeldungen des *opsiclientd* ausgegeben als auch Dialoge, die zur Steuerung des *opsiclientd* dienen. Die jeweilige Funktion und das Erscheinungsbild wird hierbei über Konfigurations-Dateien bestimmt.

Der *opsiclientd notifier* kann zu unterschiedlichen Situationen und auf unterschiedliche Weise erscheinen:

blocklogin notifier
Wird angezeigt während der *opsi-Loginblocker* aktiv ist.

Abbildung 41: opsiclientd blocklogin notifier

event notifier
Startet beim Auftreten eines Events, gibt Informationen zum Event-Ablauf aus und bietet die Möglichkeit, die Bearbeitung eines Events abzubrechen.

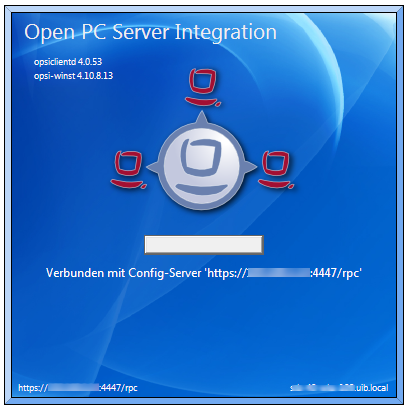


Abbildung 42: opsiclientd event notifier

action notifier

Wird gestartet wenn Aktionen ausgeführt werden sollen und bietet die Möglichkeit, diese zu verschieben.

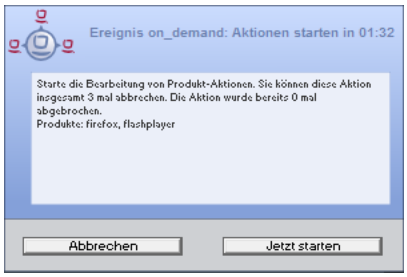


Abbildung 43: opsiclientd action notifier

shutdown notifier

Startet sobald ein Shutdown/Reboot ausgeführt werden muss und bietet die Möglichkeit, diesen zu verschieben.

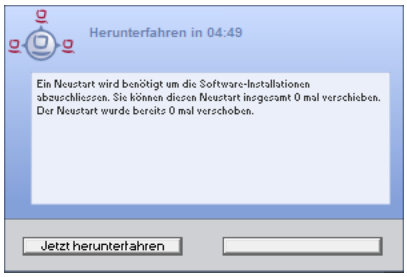


Abbildung 44: opsiclientd shutdown notifier

Achtung

Änderung der Benennung/Funktionalität von opsi 4.0.1 gegenüber opsi 4.0.
Den opsi 4.0 event notifier gibt es nicht mehr.
Der opsi 4.0.1 event notifier entspricht dem opsi 4.0 action notifier.
Der opsi 4.0.1 action notifier ähnelt dem opsi 4.0 event notifier, wird aber nur aktiv, wenn die Bearbeitung von Aktionen ansteht.

7.3.4 opsi-Loginblocker

Der *opsi-Loginblocker* für NT5 Win2K/WinXP ist als *GINA* implementiert (*opsigina.dll*). Diese *GINA* wartet bis zum Abschluss der *Produkt-Aktionen* oder dem Timeout (Standard-Wert: 120 Sekunden) bei nicht erreichbarem *opsiclientd*. Danach wird die Kontrolle an die nächste *GINA* übergeben (in der Regel an die msgina.dll).

Der *opsi-Loginblocker* für NT6 (Vista/Win7) ist als *credential provider filter* realisiert (*OpsiLoginBlocker.dll*). Er blockiert alle *credential provider* bis zum Abschluss der *Produkt-Aktionen* oder dem Timeout (Standard-Wert: 120 Sekunden) bei nicht erreichbarem *opsiclientd*.

7.3.5 Event-Ablauf

Der Ablauf der Aktionen, die in einem Event stattfinden, ist vielfältig konfigurierbar. Um die Konfigurations-Möglichkeiten zu verstehen, ist ein Verständnis der Ablauf-Logik notwendig. Es folgt zunächst ein Überblick über den Ablauf eines "Standard-Events" bei dem der opsi-configserver gefragt wird, ob Aktionen auszuführen sind (z.B. *event_gui_startup*).

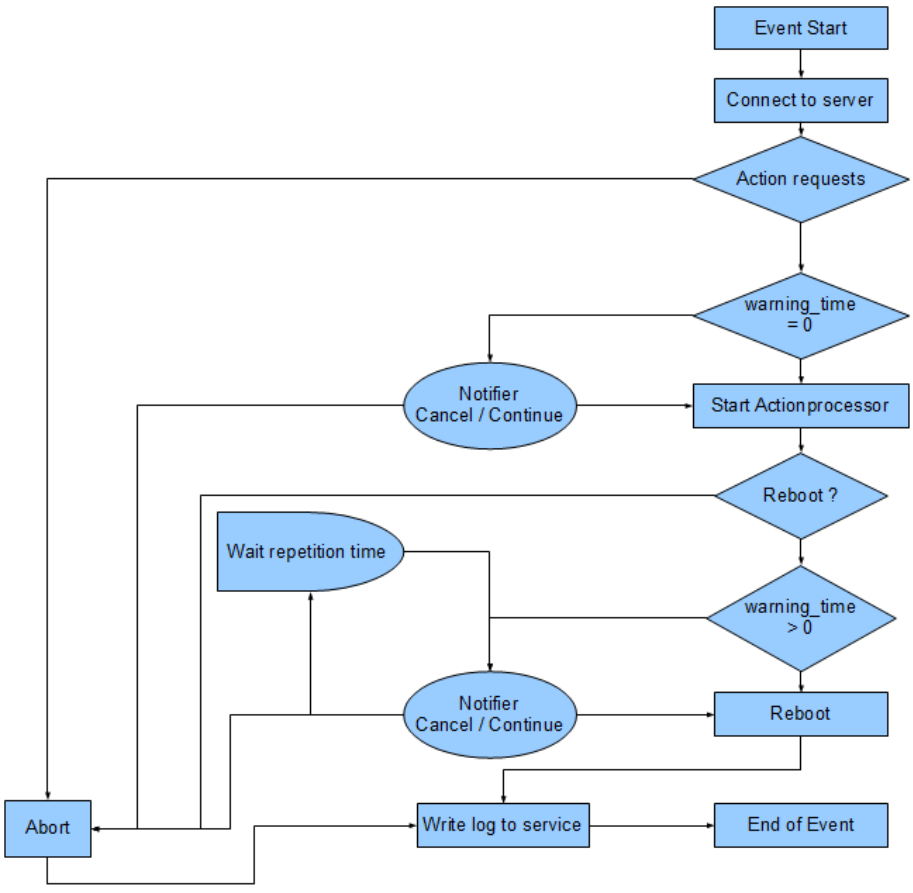


Abbildung 45: Ablauf eines Standard-Events

Die wichtigsten Parameter wirken hier wie folgt zusammen:

Tipp
Tritt bei der Verbindungsaufnahme zum *opsi-configserver* ein Fehler auf, kann natürlich auch keine Log-Datei zum *opsi-configserver* übertragen werden. Die genaue Fehlerbeschreibung ist jedoch in der `opsiclientd.log` im Log-Verzeichnis auf dem Client festgehalten.

1. Tritt ein Event ein, wird der `event_notifier_command` ausgeführt.
   Nun wird versucht die konfigurierten *opsi-configserver* über deren URLs zu erreichen.
   Konnte nach `user_cancelable_after` Sekunden keine Verbindung hergestellt werden, so wird im *opsiclientd notifier* der Button aktiviert, der das Abbrechen der Verbindungsaufnahme ermöglicht. Sobald die Verbindung zum *opsi-configserver* hergestellt ist, ist ein Abbrechen nicht mehr möglich.
   Kann innerhalb von `connection_timeout` Sekunden keine Verbindung zum *opsi-configserver* hergestellt werden, so wird das laufende Event mit einem Fehler beendet. Soll der User keine Möglichkeit zum Abbrechen haben, muss `user_cancelable_after` auf einen Wert größer oder gleich `connection_timeout` gesetzt werden.

2. Wird der *opsi-configserver* erreicht, wird geprüft, ob Aktionen gesetzt sind. Sollen Aktionen ausgeführt werden wird der `action_notifier_command` ausgeführt.
   Dieser *opsiclientd notifier* zeigt die Liste der Produkte an, für die Aktionen gesetzt sind und ist `action_warning_time` Sekunden sichtbar. Ist die `action_warning_time` = 0 (Standard-Wert) wird kein `action_notifier_command` ausgeführt.
   Zusätzlich kann dem Anwender ermöglicht werden, das Bearbeiten der Aktionen auf einen späteren Zeitpunkt zu verschieben. Die Aktionen können hierbei `action_user_cancelable` mal verschoben werden.
   Nach Erreichen der maximalen Abbrüche oder im Fall von `action_user_cancelable` = 0 kann der Anwender die Aktionen nicht verhindern.
   In jedem Fall wird ein Button angezeigt, mit dem die Wartezeit abgebrochen und die Bearbeitung der Aktionen ohne weitere Verzögerung begonnen werden kann. Der Hinweis-Text, der im *opsiclientd notifier* erscheint, ist über die Option `action_message` bzw `action_message[lang]` konfigurierbar.
   Innerhalb dieses Textes können die Platzhalter `%action_user_cancelable%` (Gesamtanzahl der möglichen Abbrüche) und `%action_cancel_counter%` (Anzahl der bereits erfolgten Abbrüche) verwendet werden.
   Wurden die Aktionen nicht vom User abgebrochen, wird der `action_cancel_counter` zurückgesetzt und der *opsi-winst* startet mit deren Bearbeitung.

3. Beendet sich der *opsi-winst* mit einer Reboot-/Shutdown-Anforderung so wird geprüft ob ein `shutdown_notifier_command` gesetzt ist und ob sie `shutdown_warning_time` > 0 ist. Sind diese Bedingungen erfüllt, wird der `shutdown_notifier_command` ausgeführt.
   Der nun startende *opsiclientd notifier* kündigt den Reboot / Shutdown an und ist `shutdown_warning_time` Sekunden sichtbar.
   Die maximale Anzahl, wie oft ein Reboot/Shutdown vom Benutzer verschoben werden kann, wird hierbei über `shutdown_user_cancelable` konfiguriert.
   In jedem Fall bietet der *opsiclientd notifier* die Möglichkeit, den Shutdown/Reboot sofort auszuführen.
   Bei einem Verschieben der Reboot-/Shutdown-Anforderung durch den Benutzer erscheint der *opsiclientd notifier* nach `shutdown_warning_repetition_time` Sekunden wieder.
   Der Hinweis-Text ist über `shutdown_warning_message` bzw. `shutdown_warning_message[lang]` konfigurierbar.
   Innerhalb dieses Textes können die Platzhalter `%shutdown_user_cancelable%` (Gesamtanzahl der möglichen Abbrüche) und `%shutdown_cancel_counter%` (Anzahl der bereits erfolgten Abbrüche) verwendet werden.
   Nach erfolgtem Shutdown oder Reboot wird der `shutdown_cancel_counter` zurückgesetzt.

Tipp
Der Ablauf des Event und auch die Aktionen des Benutzers sind in der Timeline auf der Info-Seite des *opsiclientds* sichtbar (siehe Abschnitt 7.3.8).

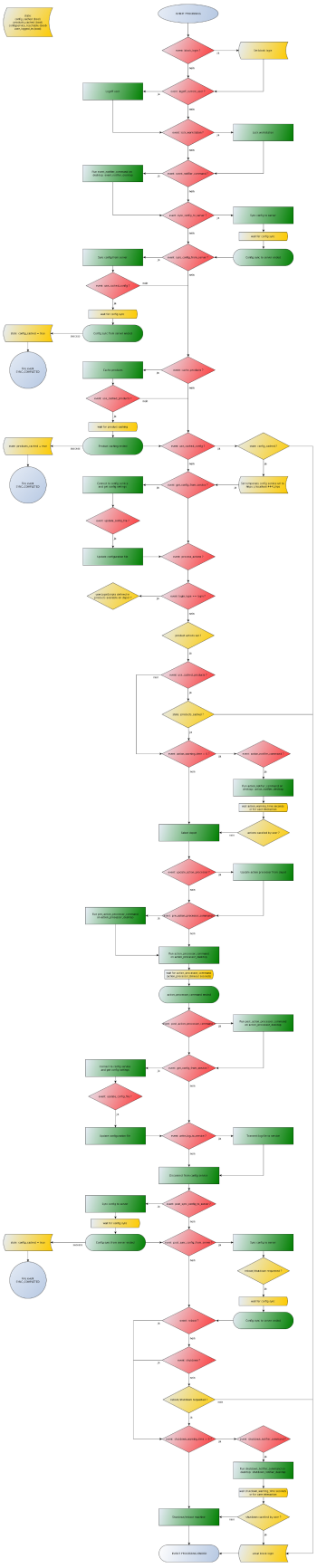


Abbildung 46: Vollständiges Ablaufdiagramm eines Events

7.3.6 Konfiguration

Konfiguration unterschiedlicher Events

Um den vielen unterschiedlichen Situationen gerecht zu werden, in denen der *opsi-client-agent* aktiv werden kann, sind die Konfigurations-Möglichkeiten vielfältig.
In der Konfiguration des *opsiclientd* leitet eine Sektion in der Form `[event_<config-id>]` eine neue Event-Konfiguration ein.
Eine Event-Konfiguration kann über das Setzen der Option `active = false` deaktiviert werden. Existiert zu einem Event-Typ keine Event-Konfiguration (oder sind diese deaktiviert), wird der entsprechende Event-Typ komplett deaktiviert.
Es gibt verschiedene Typen von Event-Konfigurationen (`type`).

- Es gibt *Event-Konfigurations-Vorlagen* (type = template)
  Event-Konfigurationen können voneinander "erben". Ist über die Option `super` die Id einer anderen Event-Konfiguration gesetzt, erbt die Event-Konfiguration alle Optionen (bis auf `active`) der Parent-Konfiguration. Geerbte Optionen können jedoch überschrieben werden.
  Das Deaktivieren von Events beeinflusst die Vererbung nicht.

- Alle weiteren Event-Konfigurationen gelten für einen gewissen Event-Typ (type).
  Verfügbare Event-Typen sind:

  - `gui startup`
    Ein Event vom Typ `gui startup` tritt beim Start des Clients (der GUI) auf.
    Es ist das gängigste Event und ist in der Standard-Konfiguration aktiv.
  - `custom`
    Event-Konfigurationen vom Typ `custom` können selbst festlegen, wann ein solches Event erzeugt wird. Hierfür kann über die Option `wql` ein *WQL*-Ausdruck angegeben werden. Sobald dieser *WQL*-Ausdruck ein Ergebnis liefert, wird ein `custom`-Event mit der jeweiligen Konfiguration gestartet.
    Wird bei einem `custom`-Event die Option `wql` leer angegeben, tritt dieses Event praktisch nie auf, kann aber über die Webservice-Schnittstelle des *opsiclientd* bei Bedarf ausgelöst werden.
  - `user login`
    Wird ausgelöst, wenn sich ein Benutzer am System anmeldet.
  - `timer`
    Tritt in festen Intervallen auf (alle `interval` Sekunden).
  - `sync completed`
    Wird ausgelöst, wenn die Synchronisation von Konfigurationen (`sync_config_from_server`) oder von Produkten (`cache_products`) erfolgt.
  - `sw on demand`
    Tritt auf, wenn ein Benutzer bei Verwendung des *Software-On-Demand-Moduls Aktionen sofort ausführen* wählt.

- Es gibt *Preconditions* (Vorbedingungen)
  Preconditions geben bestimmte Systemzustände vor (z.B. ob gerade ein Benutzer am System angemeldet ist). In der Konfiguration des *opsiclientd* leitet eine Sektion in der Form `[precondition_<precondition-id>]` die Deklaration einer *Precondition* ein. Eine *Precondition* ist dann erfüllt, wenn alle angegebenen Optionen erfüllt sind. Eine nicht angegebene Option gilt hierbei als erfüllt. Mögliche Optionen für *Preconditions* sind:

  - `user_logged_in`: ist erfüllt, wenn ein Benutzer am System angemeldet ist.
  - `config_cached`: ist erfüllt, wenn das Cachen von Konfigurationen abgeschlossen ist (siehe: `sync_config_from_server`).
  - `products_cached`: ist erfüllt, wenn das Cachen von Produkten abgeschlossen ist (siehe: `cache_products`).

- Einer Event-Konfiguration kann eine *Precondition* zugewiesen werden.
  Zu einer Event-Konfiguration mit *Precondition* muss immer eine entsprechende Event-Konfiguration ohne *Precondition* existieren. Hierbei erbt die Event-Konfiguration mit *Precondition* automatisch von der Event-Konfiguration ohne *Precondition*.

Beim Auftreten eines Events wird nun entschieden welche *Preconditions* erfüllt sind. Ist keine der *Preconditions* erfüllt, gilt die Event-Konfiguration ohne *Precondition*. Ist eine der *Preconditions* erfüllt, gilt die Event-Konfiguration die mit dieser *Precondition* verknüpft ist. Sind mehrere *Preconditions* erfüllt, so wird die *Precondition* bevorzugt, die am genauesten definiert ist (die meisten Optionen besitzt).

Ein Beispiel zur Erläuterung:
Im Rahmen einer Installation kann es notwendig sein den Rechner zu rebooten. Ist gerade ein Benutzer am System angemeldet, sollte dieser über den anstehenden Reboot informiert werden. Hierbei ist eine angemessene Wartezeit vor dem Ausführen des Reboots angebracht. Zusätzlich kann es sinnvoll sein, dem Benutzer die Entscheidung zu überlassen, ob der Reboot besser zu einem späteren Zeitpunkt ausgeführt werden soll.
Ist zum Zeitpunkt des benötigten Reboots jedoch kein Benutzer angemeldet, ist es sinnvoll, den Reboot ohne weitere Wartezeit sofort durchzuführen.
Dieses Problem wird am Beispiel von `event_on_demand` wie folgt konfiguriert:

- Es wird eine *Precondition* `user_logged_in` definiert, die erfüllt ist, wenn ein Benutzer am System angemeldet ist (`user_logged_in = true`).
- In der Event-Konfiguration `event_on_demand` (ohne *Precondition*) wird `shutdown_warning_time = 0` gesetzt (sofortiger Reboot ohne Meldung).
- In der Event-Konfiguration `event_on_demand{user_logged_in}` wird `shutdown_warning_time = 300` gesetzt (300 Sekunden Vorwarnzeit).

Konfiguration über die Konfigurationsdatei

Die Konfigurationsdatei ist:
`c:\program files\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf`

Achtung
Diese Konfigurationsdatei ist UTF-8 kodiert.
Änderungen mit Editoren, die diese Kodierung nicht beherrschen (z.B. notepad.exe), zerstören die Umlaute in dieser Datei.

Die hier festgelegte Konfiguration kann nach erfolgreicher Verbindung zum *opsi-configserver* durch die dort festgelegte *Host-Parameter* überschrieben werden. Beispiel `opsiclientd.conf`:

```
; = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
; =     configuration file for opsiclientd                              =
; = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =


; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     global settings                                                 -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[global]

# Location of the log file.
log_file = c:\\tmp\\opsiclientd.log

# Set the log (verbosity) level
# (0 <= log level <= 9)
# 0: nothing, 1: essential, 2: critical, 3: errors, 4: warnings, 5: notices
# 6: infos, 7: debug messages, 8: more debug messages, 9: passwords
log_level = 4

# Client id.
host_id =

# Opsi host key.
```

```
opsi_host_key =

# Verify opsi server certs
verify_server_cert = false

# Verify opsi server certs by ca
verify_server_cert_by_ca = false

# On every daemon startup the user login gets blocked
# If the gui starts up and no events are being processed the login gets unblocked
# If no gui startup is noticed after <wait_for_gui_timeout> the login gets unblocked
# Set to 0 to wait forever
wait_for_gui_timeout = 120

# Application to run while blocking login
block_login_notifier = %global.base_dir%\\notifier.exe -s notifier\\block_login.ini

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     config service settings                                         -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[config_service]
# Service url.
# http(s)://<opsi config server address>:<port>/rpc
url = https://opsi.uib.local:4447/rpc

# Conection timeout.
connection_timeout = 30

# The time in seconds after which the user can cancel the connection establishment
user_cancelable_after = 30

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     depot server settings                                           -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[depot_server]

# Depot server id
depot_id =

# Depot url.
# smb://<depot address>/<share name>/<path to products>
url =

# Local depot drive
drive =

# Username that is used for network connection [domain\]<username>
username = pcpatch

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     cache service settings                                          -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[cache_service]
# Maximum product cache size in bytes
product_cache_max_size = 5000000000

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     control server settings                                         -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[control_server]

# The network interfaces to bind to.
# This must be the IP address of an network interface.
# Use 0.0.0.0 to listen to all interfaces
interface = 0.0.0.0

# The port where opsiclientd will listen for HTTPS rpc requests.
port = 4441
```

```
# The location of the server certificate.
ssl_server_cert_file = %global.base_dir%\\opsiclientd\\opsiclientd.pem

# The location of the server private key
ssl_server_key_file = %global.base_dir%\\opsiclientd\\opsiclientd.pem

# The location of the static files
static_dir = %global.base_dir%\\opsiclientd\\static_html

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     notification server settings                                    -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[notification_server]

# The network interfaces to bind to.
# This must be the IP address of an network interface.
# Use 0.0.0.0 to listen to all interfaces
interface = 127.0.0.1

# The first port where opsiclientd will listen for notification clients.
start_port = 44000

# Port for popup notification server
popup_port = 45000

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     opsiclientd notifier settings                                   -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[opsiclientd_notifier]

# Notifier application command
command = %global.base_dir%\\notifier.exe -p %port% -i %id%

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     opsiclientd rpc tool settings                                   -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[opsiclientd_rpc]

# RPC tool command
command = %global.base_dir%\\opsiclientd_rpc.exe "%global.host_id%" "%global.opsi_host_key%" "%control_server.port%"

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     action processor settings                                       -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[action_processor]
# Locations of action processor
local_dir = %global.base_dir%\\opsi-winst
remote_dir = opsi-winst\\files\\opsi-winst
filename = winst32.exe

# Action processor command
command = "%action_processor.local_dir%\\%action_processor.filename%" /opsiservice "%service_url%" /clientid %global.\
     host_id% /username %global.host_id% /password %global.opsi_host_key%

# Load profile / environment of %run_as_user%
create_environment = false

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     events                                                          -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[event_default]
; === Event configuration
# Type of the event (string)
type = template
# Interval for timer events in seconds (int)
interval = -1
# Maximum number of event repetitions after which the event will be deactivated (int, -1 = forever)
```

```
max_repetitions = -1
# Time in seconds to wait before event becomes active (int, 0 to disable delay)
activation_delay = 0
# Time in seconds to wait before an event will be fired (int, 0 to disable delay)
notification_delay = 0
# Event notifier command (string)
event_notifier_command = %opsiclientd_notifier.command% -s notifier\\event.ini
# The desktop on which the event notifier will be shown on (current/default/winlogon)
event_notifier_desktop = current
# Block login while event is been executed (bool)
block_login = false
# Lock workstation on event occurrence (bool)
lock_workstation = false
# Logoff the current logged in user on event occurrence (bool)
logoff_current_user = false
# Get config settings from service (bool)
get_config_from_service = true
# Store config settings in config file (bool)
update_config_file = true
# Transmit log file to opsi service after the event processing has finished (bool)
write_log_to_service = true
# Shutdown machine after action processing has finished (bool)
shutdown = false
# Reboot machine after action processing has finished (bool)
reboot = false

; === Sync/cache settings
# Sync configuration from local config cache to server (bool)
sync_config_to_server = false
# Sync configuration from server to local config cache (bool)
sync_config_from_server = false
# Sync configuration from local config cache to server after action processing (bool)
post_sync_config_to_server = false
# Sync configuration from server to local config cache after action processing (bool)
post_sync_config_from_server = false
# Work on local config cache
use_cached_config = false
# Cache products for which actions should be executed in local depot cache (bool)
cache_products = false
# Maximum transfer rate when caching products in byte/s (int, 0 = no limit)
cache_max_bandwidth = 0
# Dynamically adapt bandwith to other network traffic (bool)
cache_dynamic_bandwidth = false
# Work on local depot cache
use_cached_products = false

; === Action notification (if product actions should be processed)
# Time in seconds for how long the action notification is shown (int, 0 to disable)
action_warning_time = 0
# Action notifier command (string)
action_notifier_command = %opsiclientd_notifier.command% -s notifier\\action.ini
# The desktop on which the action notifier will be shown on (current/default/winlogon)
action_notifier_desktop = current
# Message shown in the action notifier window (string)
action_message = Starting to process product actions. You are allowed to cancel this event a total of %\
    action_user_cancelable% time(s). The event was already canceled %state.action_processing_cancel_counter% time(s).
# German translation (string)
action_message[de] = Starte die Bearbeitung von Produkt-Aktionen. Sie können diese Aktion insgesamt %\
    action_user_cancelable% mal abbrechen. Die Aktion wurde bereits %state.action_processing_cancel_counter% mal \
    abgebrochen.
# Number of times the user is allowed to cancel the execution of actions (int)
action_user_cancelable = 0

; === Action processing
# Should action be processed by action processor (bool)
process_actions = true
# Type of action processing (default/login)
action_type = default
```

```
# Update the action processor from server before starting it (bool)
update_action_processor = true
# Command which should be executed before start of action processor
pre_action_processor_command =
# Action processor command (string)
action_processor_command = %action_processor.command%
# The desktop on which the action processor command will be started on (current/default/winlogon)
action_processor_desktop = current
# Action processor timout in seconds (int)
action_processor_timeout = 10800
# Command which should be executed before after action processor has ended
post_action_processor_command =

; === Shutdown notification (if machine should be shut down or rebooted)
# Process shutdown requests from action processor
process_shutdown_requests = true
# Time in seconds for how long the shutdown notification is shown (int, 0 to disable)
shutdown_warning_time = 0
# Shutdown notifier command (string)
shutdown_notifier_command = %opsiclientd_notifier.command% -s notifier\\shutdown.ini
# The desktop on which the action notifier will be shown on (current/default/winlogon)
shutdown_notifier_desktop = current
# Message shown in the shutdown notifier window (string)
shutdown_warning_message = A reboot is required to complete software installation tasks. You are allowed to delay this \
    reboot a total of %shutdown_user_cancelable% time(s). The reboot was already delayed %state.\
    shutdown_cancel_counter% time(s).
# German translation (string)
shutdown_warning_message[de] = Ein Neustart wird benötigt um die Software-Installationen abzuschliessen. Sie können \
    diesen Neustart insgesamt %shutdown_user_cancelable% mal verschieben. Der Neustart wurde bereits %state.\
    shutdown_cancel_counter% mal verschoben.
# Number of times the user is allowed to cancel the shutdown (int)
shutdown_user_cancelable = 0
# Time in seconds after the shutdown notification will be shown again after the user has canceled the shutdown (int)
shutdown_warning_repetition_time = 3600

[event_gui_startup]
super = default
type = gui startup
name = gui_startup
block_login = true

[event_gui_startup{user_logged_in}]
name = gui_startup
shutdown_warning_time = 300
block_login = false

[event_gui_startup{cache_ready}]
use_cached_config = true
use_cached_products = true
action_user_cancelable = 3
action_warning_time = 60

[event_on_demand]
super = default
type = custom
name = on_demand

[event_on_demand{user_logged_in}]
name = on_demand
shutdown_warning_time = 300

[event_software_on_demand]
super = default
type = sw on demand

[event_sync]
super = default
type = template
```

```
process_actions = false
event_notifier_command =
sync_config_to_server = true
sync_config_from_server = true
cache_products = true
cache_dynamic_bandwidth = true

[event_timer]
super = sync
type = timer
active = false
interval = 3600

[event_net_connection]
super = sync
type = custom
active = false
wql = SELECT * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_NetworkAdapter' AND \
     TargetInstance.NetConnectionStatus = 2

[event_sync_completed]
super = default
type = sync completed
event_notifier_command =
process_actions = false
get_config_from_service = false
write_log_to_service = false

[event_sync_completed{cache_ready_user_logged_in}]
reboot = true
shutdown_user_cancelable = 10
shutdown_warning_time = 300

[event_sync_completed{cache_ready}]
reboot = true

[event_user_login]
super = default
type = user login
action_type = login
active = false
message = Starting to process user login actions.
message[de] = Beginne mit der Verarbeitung der Benutzer-Anmeldungs-Aktionen.
block_login = false
process_shutdown_requests = false
get_config_from_service = false
update_config_file = false
write_log_to_service = false
update_action_processor = false
action_notifier_command = %opsiclientd_notifier.command% -s notifier\\userlogin.ini
action_notifier_desktop = default
action_processor_command = %action_processor.command% /usercontext %event.user%
action_processor_desktop = default
action_processor_timeout = 300

[precondition_user_logged_in]
user_logged_in = true

[precondition_cache_ready]
config_cached = true
products_cached = true

[precondition_cache_ready_user_logged_in]
user_logged_in = true
config_cached = true
products_cached = true
```

Konfiguration über den Webservice (Host-Parameter)

Die Konfiguration kann auch zentral gesteuert werden. Hierzu dienen Einträge in der *Host-Parameter* des *opsi-configservers*.

Diese Einträge müssen dem folgenden Muster folgen:
`opsiclientd.<name der section>.<name der option>`

Ein Beispiel:
`opsiclientd.event_gui_startup.action_warning_time = 20`
setzt in der Konfigurationsdatei `opsiclientd.conf` in der Sektion `[event_gui_startup]` den Wert von `action_warning_time` auf 20.

Die folgende Abbildung zeigt, wie diese Werte als Defaults für alle Clients über den *opsi-configed* gesetzt werden können.

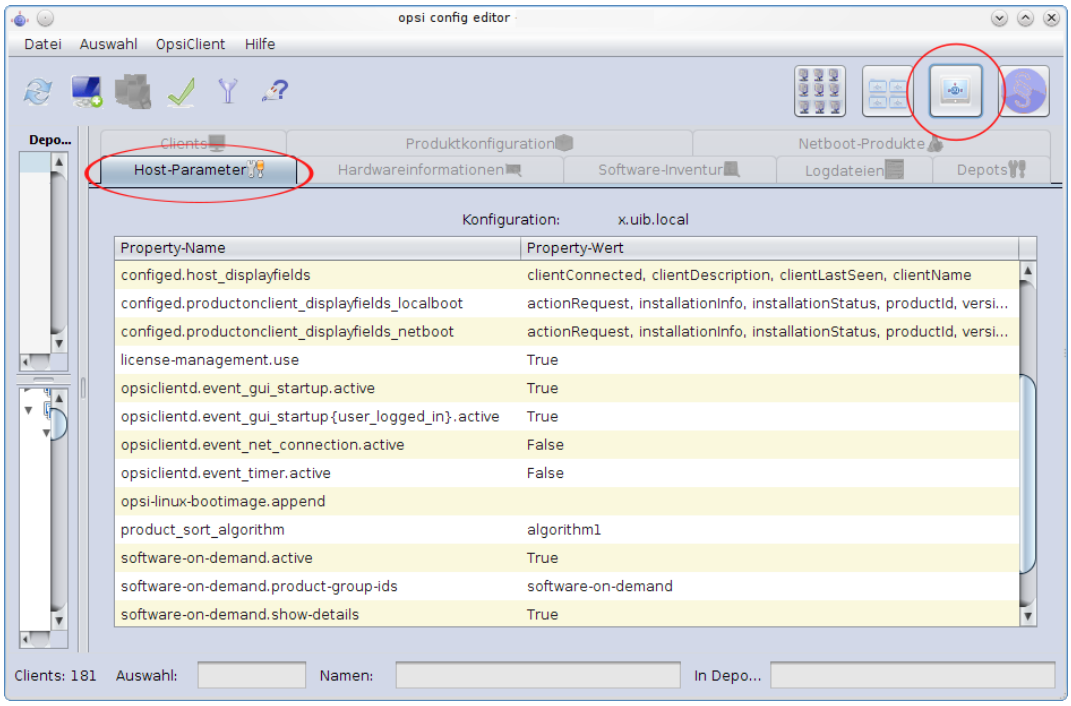


Abbildung 47: Serverweite Konfiguration des opsiclientd über den opsi-configed

Hier kann über das Kontextmenü `Property hinzufügen` ein neuer Wert gesetzt werden.

Um einen Host-Parameter zu löschen, verwenden Sie das Werkzeug *opsi-admin*. Beispiel:

```
opsi-admin -d method config_delete "opsiclientd.event_gui_startup.action_warning_time"
```

Eine Client-spezifische Änderung über den *opsi-configed* führen Sie über den *Hosts-Parameter* Tab in der Client-Konfiguration aus. Um Client-spezifische Einträge zu löschen, verwenden Sie das Werkzeug *opsi-admin*. Beispiel:

```
@opsi-admin> method configState_delete "opsiclientd.event_gui_startup.action_warning_time" "myclient.uib.local"
```

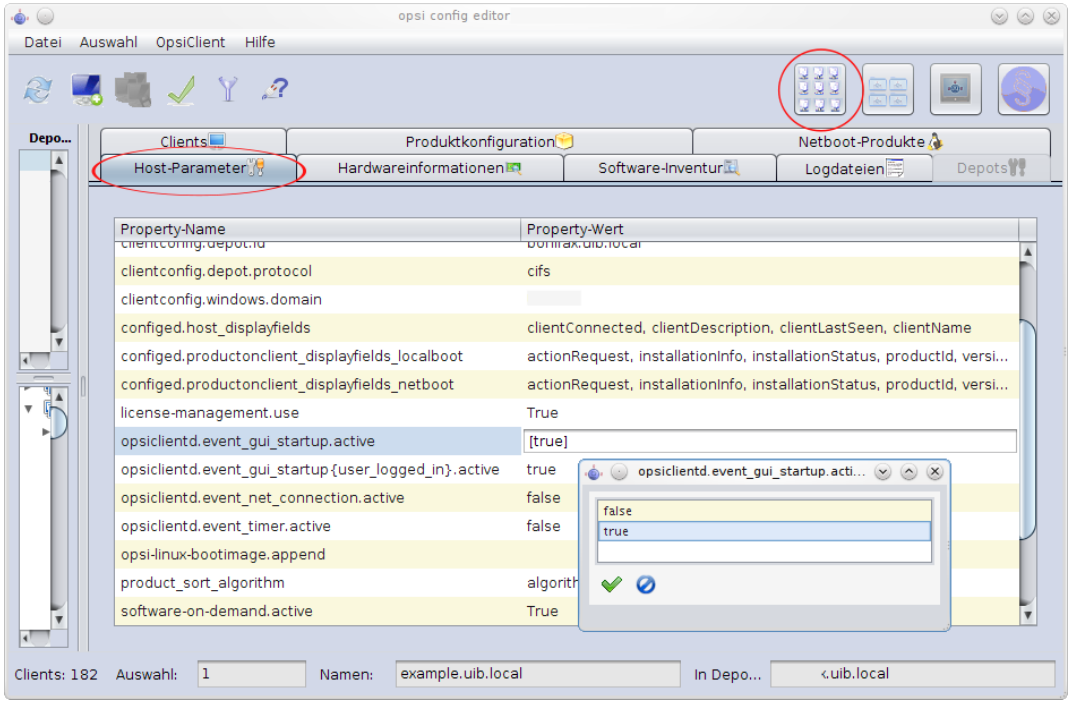


Abbildung 48: Client-spezifische Konfiguration des opsiclientd über den opsi-configed

7.3.7 Logging

Die Log-Datei des *opsiclientd* ist standardmäßig `c:\tmp\opsiclientd.log`.

Die Log-Informationen werden auch an den *opsi-configserver* übertragen. Dort liegen sie unter */var/log/opsi/clientconnect/<ip-bzw.-name-des-clients>.log*. Sie sind auch im *opsi-configed* über Logdateien ⇒ Clientconnect einsehbar.

Jede Zeile in der Logdatei folgt dem Muster:
`[<log level>] [<datum zeit>] [Quelle der Meldung] Meldung (Quellcode-Datei|Zeilennummer).`

Dabei gibt es die folgenden Log-Level:

```
# Set the log (verbosity) level
# (0 <= log level <= 9)
# 0: nothing, 1: essential, 2: critical, 3: errors, 4: warnings, 5: notices
# 6: infos, 7: debug messages, 8: more debug messages, 9: passwords
```

Beispiel:

```
(...)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'sync_completed{cache_ready}' added to event \
    generator 'sync_completed'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'gui_startup' added to event generator '\
    gui_startup'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'gui_startup{cache_ready}' added to event \
    generator 'gui_startup'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'on_demand' added to event generator 'on_demand'  \
     (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'sync_completed{cache_ready_user_logged_in}' added\
     to event generator 'sync_completed'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'gui_startup{user_logged_in}' added to event \
    generator 'gui_startup'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'sync_completed' added to event generator '\
    sync_completed'   (Events.pyo|1107)
```

```
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'software_on_demand' added to event generator '\
    software_on_demand'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Event config 'on_demand{user_logged_in}' added to event \
    generator 'on_demand'   (Events.pyo|1107)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] Updating config file: 'C:\Program Files (x86)\opsi.org\opsi-\
    client-agent\opsiclientd\opsiclientd.conf'   (Config.pyo|287)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] No need to write config file 'C:\Program Files (x86)\opsi.org\\
    opsi-client-agent\opsiclientd\opsiclientd.conf', config file is up to date   (Config.pyo|318)
[5] [Mar 22 10:17:46] [ event processing gui_startup  ] No product action requests set   (EventProcessing.pyo|591)
[5] [Mar 22 10:17:49] [ event processing gui_startup  ] Writing log to service   (EventProcessing.pyo|247)
[6] [Mar 22 10:17:49] [ opsiclientd                   ] shutdownRequested: 0   (Windows.pyo|340)
[6] [Mar 22 10:17:49] [ opsiclientd                   ] rebootRequested: 0   (Windows.pyo|326)
[5] [Mar 22 10:17:49] [ opsiclientd                   ] Block login now set to 'False'   (Opsiclientd.pyo|111)
[6] [Mar 22 10:17:49] [ opsiclientd                   ] Terminating block login notifier app (pid 1620)   (Opsiclientd.\
    pyo|148)
[6] [Mar 22 10:17:49] [ event processing gui_startup  ] Stopping notification server   (EventProcessing.pyo|225)
[6] [Mar 22 10:17:51] [ control server                ] client connection lost   (Message.pyo|464)
[6] [Mar 22 10:17:52] [ event processing gui_startup  ] Notification server stopped   (Message.pyo|651)
[5] [Mar 22 10:17:52] [ event processing gui_startup  ] ============= EventProcessingThread for event 'gui_startup' \
    ended =============   (EventProcessing.pyo|1172)
[5] [Mar 22 10:17:52] [ opsiclientd                   ] Done processing event '<ocdlib.Events.GUIStartupEvent object at\
     0x023CE330>'   (Opsiclientd.pyo|405)
[5] [Mar 22 10:19:41] [ opsiclientd                   ] Session 'HSzMB1wtOiBS6vHl7mh3ro5r6s3TanFu' from ip '127.0.0.1',\
     application 'opsi jsonrpc module version 4.0.1' expired after 120 seconds   (Session.pyo|184)
[6] [Mar 22 10:19:41] [ opsiclientd                   ] Session timer <_Timer(Thread-20, started daemon 2636)> canceled\
       (Session.pyo|120)
[5] [Mar 22 10:19:41] [ opsiclientd                   ] Session 'HSzMB1wtOiBS6vHl7mh3ro5r6s3TanFu' from ip '127.0.0.1',\
     application 'opsi jsonrpc module version 4.0.1' deleted   (Session.pyo|207)
[6] [Mar 22 10:27:55] [ control pipe                  ] Creating pipe \\.\pipe\opsiclientd   (ControlPipe.pyo|253)
[5] [Mar 22 10:27:55] [ event generator wait_for_gui  ] -----> Executing: getBlockLogin()   (JsonRpc.pyo|123)
[5] [Mar 22 10:27:55] [ opsiclientd                   ] rpc getBlockLogin: blockLogin is 'False'   (ControlPipe.pyo\
    |428)
[6] [Mar 22 10:27:55] [ event generator wait_for_gui  ] Got result   (JsonRpc.pyo|131)
'
```

Die Log-Datei des *opsi-Loginblockers* befindet sich unter NT6 (Vista/Win7) als auch unter NT5 (Win2k/WinXP) in `c:\tmp\opsi_loginblocker.log`.

7.3.8 opsiclientd infopage

Da bei den Abläufen im *opsiclientd* vielfältige Komponenten zusammenwirken, welche zum Teil gleichzeitig aktiv sind, wird die Logdatei leicht unübersichtlich.

Daher verfügt der *opsiclientd* über eine eigene *infopage* welche die Abläufe auf einer Zeitachse grafisch darstellt. Diese *infopage* kann mit dem Browser über die URL https://<adresse-des-clients>:4441/info.html aufgerufen werden.

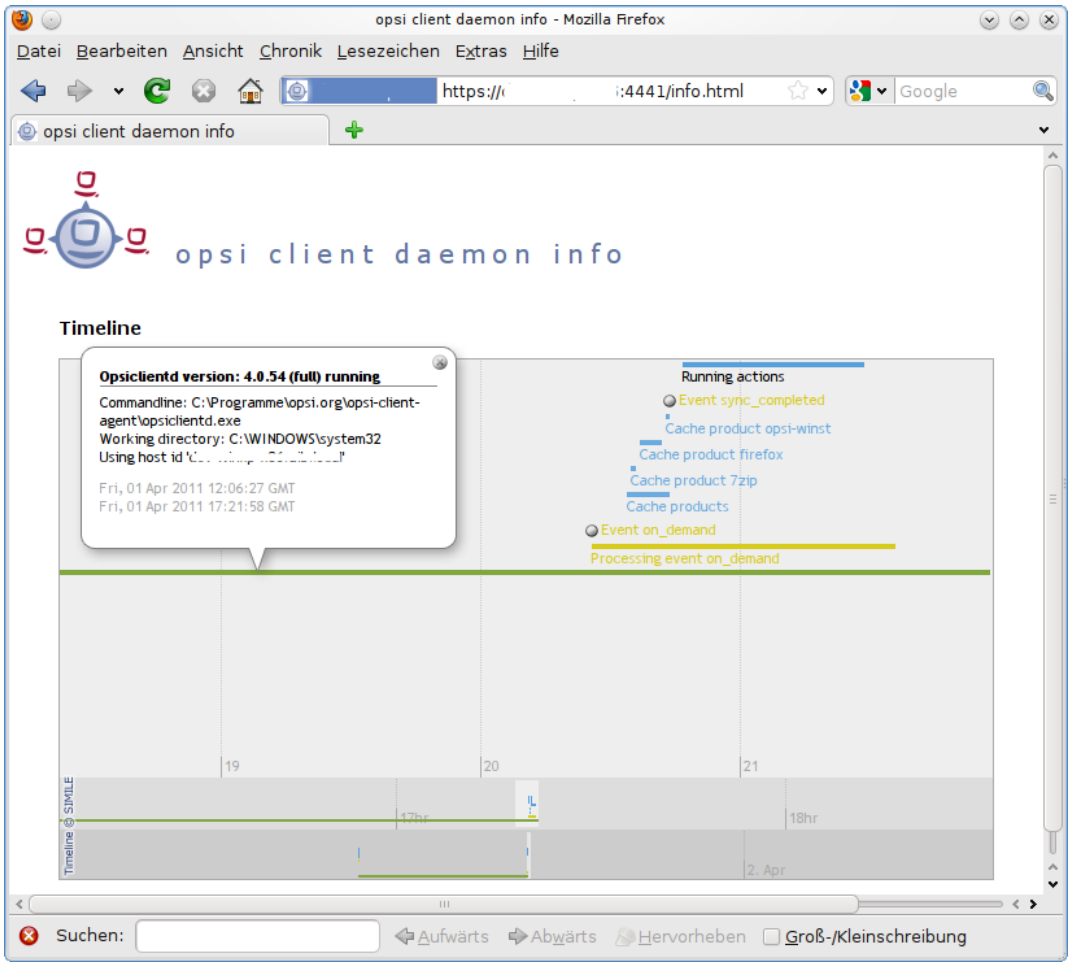


Abbildung 49: Info-Page des opsiclientd nach einer Push-Installation mit aktiviertem Produkt-Caching

7.3.9 Fernsteuerung des opsi-client-agent

Der *opsiclientd* verfügt über eine Webservice-Schnittstelle. Diese ermöglicht es, dem opsi-client-agent Anweisungen zu übermitteln und Vieles mehr. Sie lassen sich momentan grob in drei Bereiche aufteilen:

- Nachrichten (Popup) versenden
- *Push*-Installationen durch auslösen von Events (z.B. *on_demand*)
- Sonstige Wartungsarbeiten

Dies kann auch auf der Kommandozeile mittels Aufrufs einer *hostControl_**-Methode über *opsi-admin* geschehen. Bei Verwendung der *hostControl_**-Methoden
`opsi-admin -d method hostControl_xx *hostIds` kann der Parameter `*hostIds`

- entfallen, dann gilt der Aufruf für alle Clients
- einen Client enthalten (z.B. "myclient.uib.local")
- eine Liste von Clients enthalten ["<client1>", "<client2>", ...]
  z.B. ["client1.uib.local", "client2.uib.local"]
- eine Wildcard enthalten, wobei * als Platzhalter dient
  z.B. "client.*" oder "*.uib.*"

Werden Rechner nicht erreicht (z.B. weil sie aus sind), wird für diese Rechner eine Fehlermeldung ausgegeben.

Nachrichten per Popup senden

Über den *opsi-configed* lassen sich Nachrichten an einen oder mehrere Clients versenden.

Siehe dazu Kapitel Abschnitt 4.3.8

Auf der Kommandozeile lässt sich dies ebenfalls mittels *opsi-admin* durchführen:

```
opsi-admin -d method hostControl_showPopup message *hostid
```

Beispiel:

```
opsi-admin -d method hostControl_showPopup "Ein Text..." "myclient.uib.local"
```

Push-Installationen: Event *on demand* auslösen

Vom *opsi-server* aus kann der Client aufgefordert werden, die gesetzten *Produkt-Aktionen* auszuführen.

Das Auslösen des Events kann vom *opsi-configed* aus erfolgen. Abschnitt 4.3.8

Auf der Kommandozeile lässt sich dies ebenfalls mittels *opsi-admin* durchführen:

```
opsi-admin -d method hostControl_fireEvent event *hostIds
```

Beispiel:

```
opsi-admin -d method hostControl_fireEvent "on_demand" "myclient.uib.local"
```

Sonstige Wartungsarbeiten (shutdown, reboot, . . .)

Über den Webservice des *opsiclientd* ist es möglich, steuernd auf den *opsi-client-agent* einzuwirken. Dazu muss man sich an diesem Webservice authentifizieren. Dies geschieht entweder mittels des lokalen Administrator-Accounts (ein leeres Passwort ist unzulässig) oder mittels der *opsi-host-Id* (FQDN / vollständiger Host-Name inkl. DNS-Domain) als Benutzername und des *opsi-host-Schlüssels* als Passwort.

Vom *opsi-configed* aus geht dies über das Menü *OpsiClient* oder aus dem Kontextmenü des *Client*-Tabs.

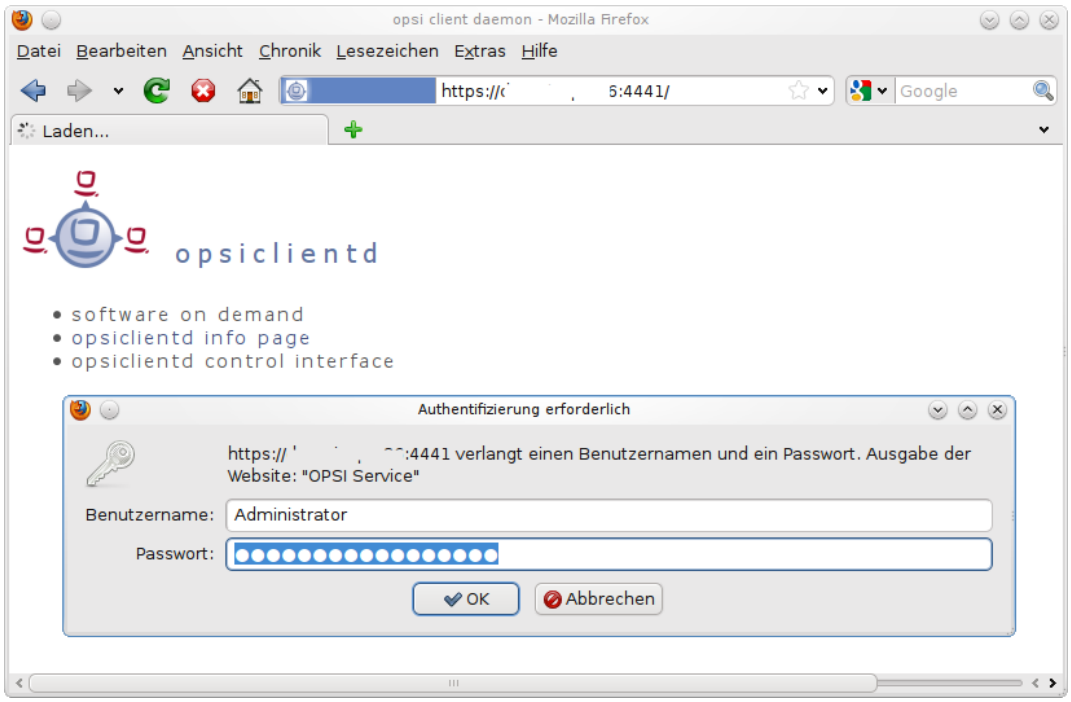


Abbildung 50: Webservice des opsiclientd

Auch auf der Kommandozeile gibt es hierfür Entsprechungen:

shutdown:

```
opsi-admin -d method hostControl_shutdown *hostIds
```

reboot:

```
opsi-admin -d method hostControl_reboot *hostIds
```

7.4 Sperrung des Anwender Logins mittels opsi-Loginblocker

Um zu verhindern, dass sich ein Anwender schon vor dem Abschluss der Installation am System anmeldet, kann zusätzlich der opsi-Loginblocker installiert werden. Dieser gibt den Zugriff auf den Login erst frei, wenn der Installations-Prozess beendet ist.

Ob der *opsi-Loginblocker* währen der *opsi-client-agent*-Installation installiert bzw. aktiviert wird, kann über das *Product-Property* `loginblockerstart` konfiguriert werden.

7.4.1 opsi-Loginblocker unter NT5 (Win2k/WinXP)

Der opsi-Loginblocker (`opsigina.dll`) ist als *GINA* realisiert. Die *opsigina* wartet bis zum Abschluss der *Produkt-Aktionen* oder dem Timeout (standard-Wert: 120 Sekunden) bei nicht erreichbarem *opsiclientd*. Danach wird die Kontrolle an die nächste *GINA* übergeben (in der Regel an die msgina.dll).

GINA steht hierbei für „Graphical Identification and Authentication“ und stellt die seitens Microsofts offiziell unterstützte Möglichkeit dar, in den Login-Prozess von Windows einzugreifen.

Gelegentlich ist es der Fall, dass bereits andere Softwareprodukte (z.B. Client für Novell-Netzwerke) eine *GINA* auf dem System installiert haben und empfindlich auf Eingriffe reagieren.

Generell sind mehrere ‚nacheinander aufgerufene *GINAs* (*GINA-chaining*) durchaus möglich. Auch die *opsigina.dll* des opsi-Loginblocker ist für das genannte *GINA-chaining* vorbereitet. Sollte der beschriebene Fall bei Ihren Clients eintreten, informieren Sie sich bitte auf dem freien Supportforum (https://forum.opsi.org) nach bestehenden Anpassungsmöglichkeiten oder kontaktieren Sie die Firma *uib*.

7.4.2 opsi-Loginblocker unter NT6 (Vista/Win7)

Der *opsi-Loginblocker* für NT6 (Vista/Win7) ist als *credential provider filter* realisiert (*OpsiLoginBlocker.dll*). Er blockiert alle *credential provider* bis zum Abschluss der *Produkt-Aktionen* oder dem Timeout (Standard-Wert: 120 Sekunden) bei nicht erreichbarem *opsiclientd*.

7.5 Nachträgliche Installation des opsi-client-agents

Die Anleitung zur nachträglichen Installation des *opsi-client-agents* finden Sie im Handbuch *opsi-getting-started* im Kapitel *Erste Schritte*.

7.5.1 Installation des opsi-client-agent in einem Master-Image oder als Exe

has to be written

8 Localboot-Produkte: Automatische Softwareverteilung mit opsi

Als Localboot-Produkte werden alle Produkte bezeichnet die nach einem lokalen Boot des Rechners über den *opsi-client-agent* installiert werden. Dies im Gegensatz zu den weiter unten beschriebenen Netboot Produkten Abschnitt 9.

8.1 opsi Standardprodukte

Die folgenden Localboot Produkte gehören zur Grundausstattung von opsi.

8.1.1 *opsi-client-agent*

Der *opsi-client-agent* ist der Clientagent von opsi und weiter oben ausführlich beschrieben.

8.1.2 *opsi-winst*

Das Produkt *opsi-winst* ist ein Spezialfall. Es enthält den aktuellen *opsi-winst*. Dieser muss zur Aktualisierung nicht auf setup gestellt werden. Vielmehr prüft ein Teil der *opsi-client-agent* bei jedem Start, ob auf dem Server eine andere Version des *opsi-winst* verfügbar ist und holt sich diese im Zweifelsfall.+ Hinweis:+ Diese Funktionalität ist unter Windows 2000 nicht gegeben. Hier muss eine neuer *opsi-winst* im Rahmen eines *opsi-client-agent* updates eingespielt werden.

8.1.3 javavm: Java Runtime Environment

Das Produkt javavm stellt die für den *opsi-configed* benötigte Java 1.6 Laufzeitumgebung für die Clients zur Verfügung.

8.1.4 *opsi-admin utils*

Das Produkt *opsi-admin utils* bietet neben einigen Hilfsprogrammen vor allem eine lokale Installation des *opsi-configed*.

8.1.5 jedit

Java basierter Editor mit Syntax Highlighting für *opsi-winst* Scripte.

8.1.6 swaudit + hwaudit: Produkte zur Hard- und Software-Inventarisierung

Die Produkte hwaudit und swaudit dienen der Hard- bzw. Software-Inventarisierung. Bei der Hardware-Inventarisierung werden die Daten über WMI erhoben und über den *opsi-webservice* an den Server zurück gemeldet. Bei der Software-Inventarisierung werden die Daten aus der Registry (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall) erhoben und über den *opsi-webservice* an den Server zurück gemeldet.

8.1.7 opsi-template

Template zur Erstellung eigener opsi-Scripts. Sie können das Template extrahieren mit

```
‘opsi-package-manager -x opsi-template_<version>.opsi‘
```

oder auch dabei gleich umbenennen mit

```
‘opsi-package-manager -x opsi-template_<version>.opsi --new-product-id myprod‘
```

8.1.8 xpconfig

Paket zum Customizing der Grundeinstellungen von Oberfläche, Explorer usw., nicht nur für XP.

8.2 Beeinflussung der Installationsreihenfolge durch Prioritäten und Produktabhängigkeiten

Seit opsi 4.0 wird die Installationsreihenfolge vom opsi-server unter Berücksichtigung von Produktabhängigkeiten und Produktprioritäten berechnet.

- Produktabhängigkeiten
  definieren Abhängigkeiten und notwendige Installationsreihenfolgen zwischen Produkt-Paketen. Typische Beispiel ist die Abhängigkeit von Java Programmen von der Java Laufzeitumgebung (javavm).
- Produktprioritäten
  dienen dazu, bestimmte Pakete in der Installationsreihenfolge nach vorne oder nach hinten zu schieben. So ist es z.B. sinnvoll Servicepacks und Patches an den Anfang einer Installation zu legen und eine Softwareinventarisierung an das Ende.
  Produktprioritäten sind Zahlen zwischen 100 und -100 (0 ist default)

Wie diese beiden Faktoren gegeneinander gewichtete werden sollen kann unterschiedlich gesehen werden. Daher stellt opsi zwei Algorithmen zur Verfügung.

Die Umstellung zwischen diesen Algorithmen erfolgt entweder:

im *opsi-configed*, in der Server-Konfiguration

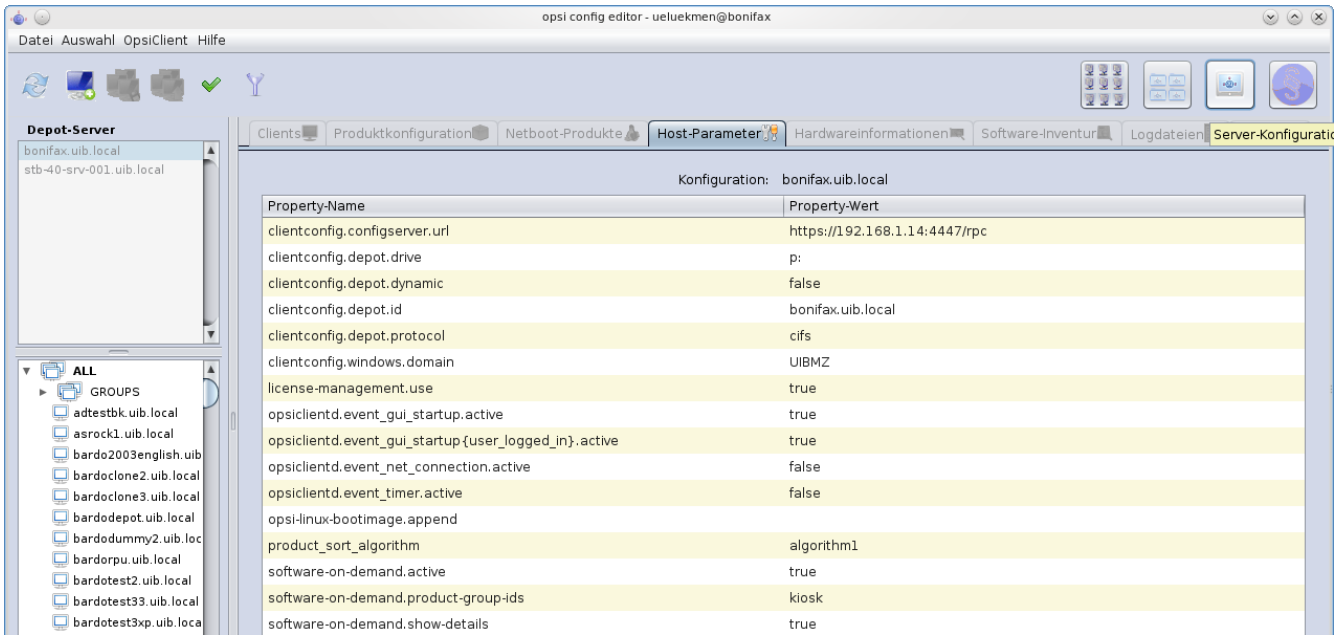


Abbildung 51: *opsi-configed*: Serverkonfiguration

oder auf der Kommandozeile mit folgendem Befehl:

```
opsi-setup --edit-config-defaults
```

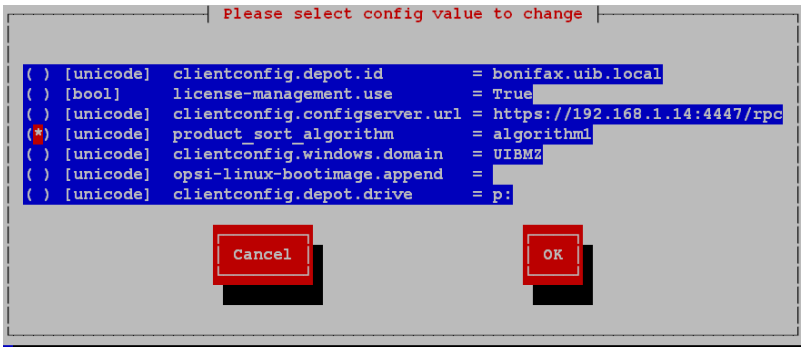


Abbildung 52: Wählen des Sortieralgorithmus: Teil 1

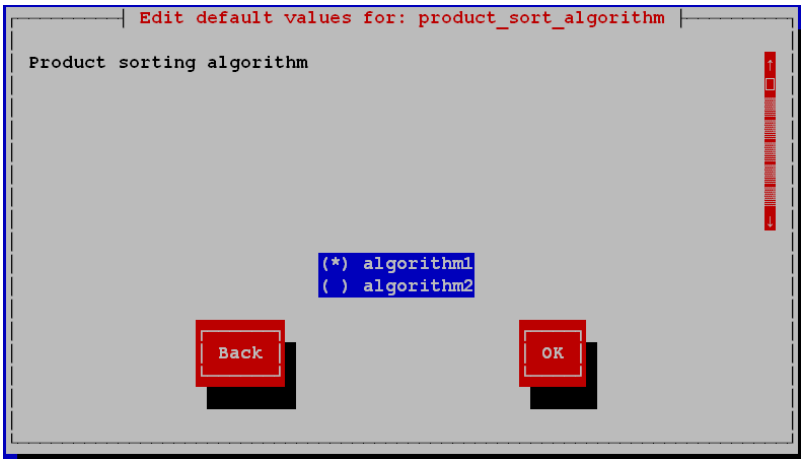


Abbildung 53: Wählen des Sortieralgorithmus: Teil 2

8.2.1 Algorithm1: Produktabhängigkeit vor Priorität (Default)

Bei diesem Algorithmus werden zunächst die Produkte anhand Ihrer Prioritäten sortiert und dann aufgrund der Produktabhängigkeiten nochmals umsortiert. Hierdurch kann natürlich ein Produkt mit sehr niedriger Priorität weit nach vorne geschoben werden weil es von einem anderen Produkt als *required before* benötigt wird. Auf der anderen Seite wird vermieden das es zu Installationsproblemen aufgrund nicht aufgelöster Produktabhängigkeiten kommt.

8.2.2 Algorithm2: Produktpriorität vor Abhängigkeit

Dieser Algorithmus geht von dem Gedanken aus, dass es in der Praxis im wesentlichen drei Prioritätsklassen gibt:

- Produkte, welche am Anfang installiert werden sollen wie z.B. OS-Patches und Treiber, durch die der PC in seinen Standard-Zustand gebracht wird. Wird realisiert durch Zuweisen einer hohen Priorität (maximal +100).
- "Normale" Produkte, die Anwendungen installieren (Default-Priorität 0).
- Produkte, die möglichst am Ende eingespielt werden sollen wie z.B. Softwareinventarisierung. Realisiert durch Zuweisen einer niedrigen Priorität (niedrigste mögliche -100).

Die Auflösung der Produktabhängigkeiten erfolgt nur innerhalb einer Prioritätsklasse. Hierdurch ist sichergestellt, dass Produkte mit einer hohen Priorität auch tatsächlich am Anfang installiert werden. Prioritätsklassen übergreifende Produktabhängigkeiten werden nicht berücksichtigt bzw. führen zu einer Warnung. Daher ist beim Packen zu beachten, dass Produktabhängigkeiten nur innerhalb einer Prioritätsklasse definiert werden.

Die Produktabhängigkeiten werden hier so interpretiert, dass sie bei "normalen" Produkten automatisch zu einer konsistenten, alle Abhängigkeiten berücksichtigenden Reihenfolge führen. Wurden widersprüchliche (zirkuläre) Abhängigkeiten definiert, wird ein Fehler angezeigt.

Bei den für die PC-Einrichtung grundlegenden Produkten mit hohen Prioritäten wird dagegen der Administrator – ähnlich wie etwa bei Unix-Startskripten – die genaue Reihenfolge von Hand festlegen, indem er pro Produkt entsprechend der gewünschten Reihenfolge je eine spezifische Priorität zwischen +100 und +1 setzt. Ähnliches gilt für die finalen Produkte mit niedrigen Prioritäten.

8.2.3 Erstellung von Prioritäten und Produktabhängigkeiten

Prioritäten und Produktabhängigkeiten gehören zu den Meta-Daten eines Produktes. Diese werden bei der Erstellung eines Produktes mit dem Befehl `opsi-newprod` abgefragt.

Diese Metadaten werden im control file des Produktes abgelegt und können dort editiert werden. Nach einer Veränderung im control file muss das Produkt neu gepackt und installiert werden.

Siehe hierzu auch das Kapitel *Erstellen eines opsi-Product-Paketes* im opsi-getting-started Handbuch.

8.3 Einbindung eigener Software in die Softwareverteilung von opsi

Die Anleitung zur Einbindung eigener Software finden Sie im Handbuch opsi-getting-started.

9 Netboot Produkte

9.1 Parametrisierung vom Linux Installationsbootimage

Das Linux Installationsbootimage hat eine Reihe von Standardeinstellungen, welche das Verhalten beeinflussen. Sollte nach dem Laden des Bootimages der Bildschirm schwarz bleiben oder die Netzwerkkarte nicht (korrekt) funktionieren, so muss evtl. für diese konkrete Hardware die Startparameter des Bootimages angepasst werden.
Dies können Sie im *opsi-configed* im Tab *Hostparameter* am Eintrag *opsi-linux-bootimage.append* tun.

Typische Werte sind hier (einzeln oder kombiniert):

- `acpi=off`
- `noapic`
- `irqpoll`
- `reboot=bios`

Eine weitere wichtige Standardeinstellung ist das Passwort von root auf dem Bootimage. Dieses ist per default *linux123* und sollte aus Sicherheitsgründen ebenfalls auf diesem Weg abgeändert werden.

Um diese angesprochenen Modifikationen durch zu führen, muss man die Konfiguration: *opsi-linux-bootimage.append* am besten in der Serverkonfiguration anpassen.

Die hier wichtige Option ist *pwh*. Dieser Option muss man das verschlüsselte Passwort als Hash-Wert mitgeben. Dieser wird dann automatisch beim Booten in die Shadow geladen. Somit wird bevor ein Login auf dem Client möglich ist, das Passwort verändert. Um diesen Hash zu bekommen gibt es verschiedene Wege. Um sicher zu gehen, dass man den richtigen Hash in der richtigen Formatierung und Art (MD5, SHA1, etc. . .) bekommt, empfehlen wir folgendes Vorgehen.

Einen opsi-Client per PXE oder mit der aktuellen opsi-clientbootcd starten. Dann per Putty oder direkt vom *opsi-server* aus per ssh als root eine Verbindung aufbauen. Vom *opsi-server* aus:

```
ssh root@<client.domain.tld>
```

Das Passwort lautet linux123. Dann einfach das Passwort von root neu setzen:

```
passwd
```

Nun muss man den gesetzten Hash holen.

```
grep root /etc/shadow
```

Die Ausgabe sollte nun folgendermaßen aussehen:

```
root:$6$344YXKIT$D4RPZfHMmv8e1/i5nNkOFaRN2oYNobCEjCHnkehiEFA7NdkDW9KF496OHBmyHHqOkD2FBLHZoTdr5YoDlIoWz\
    /:14803:0:99999:7:::
```

Um das Passwort zu setzen, reicht es wenn man den Hash kopiert, der nach dem ersten Doppelpunkt beginnt und beim zweiten Doppelpunkt in der Zeile aufhört. Die daraus resultierende Option für die Konfiguration *opsi-linux-bootimage.append* wäre:

```
pwh=$6$344YXKIT$D4RPZfHMmv8e1/i5nNkOFaRN2oYNobCEjCHnkehiEFA7NdkDW9KF496OHBmyHHqOkD2FBLHZoTdr5YoDlIoWz/
```

9.2 Automatische Betriebssysteminstallation unattended

9.2.1 Überblick

Ablauf einer Reinstallation:

- Bei PXE-Boot:
  - Über den *opsi-configed* oder *opsi-admin* wird der PC für die Neuinstallation ausgewählt.
- Der Client erkennt beim nächsten Bootvorgang mit Hilfe des PXE-Bootproms, dass er reinstalliert werden soll und lädt ein Bootimage vom *opsi-server*.
- Bei CD-Boot:
  - Der Client bootet von der opsi-clientbootcd das Bootimage.
- Das Bootimage stellt am Client die Rückfrage, ob der PC tatsächlich reinstalliert werden soll. Dies ist die einzige Interaktion des gesamten Prozesses.
- Das Bootimage partitioniert und formatiert die Festplatte.
- Das Bootimage überträgt die notwendigen Installationsdateien und Konfigurationsinformationen vom *opsi-server* auf den Client und leitet einen Reboot ein.
- Nach dem Reboot installiert der Client selbstständig das Betriebssystem anhand der übertragenen Konfigurationsinformationen.
- Im Anschluss wird der opsi-client-agent zur Einbindung der automatischen Softwareverteilung installiert.
- Die automatische Softwareverteilung installiert die gesamte Software, die gemäß Konfigurationsdatei auf diesen Rechner gehört.

9.2.2 Voraussetzungen

Der Client-PC sollte mit einer bootfähigen Netzwerkkarte ausgestattet sein. Viele heute eingesetzte Netzwerkkarten verfügen über eine entsprechende PXE-Firmware. Diese kontrolliert den Bootvorgang vom Netz, falls nicht eine andere Reihenfolge im BIOS eingestellt ist. Ist kein PXE vorhanden kann alternativ kann auch von der opsi-clientbootcd das Bootimage gebootet werden.

Das opsi-Installationspaket für das zu installierende Betriebssystem muss auf dem opsiserver installiert sein. In den folgenden Abschnitten wird als Beispiel jeweils Windows XP angenommen.

9.2.3 PC-Client bootet vom Netz

Die Firmware des PXE wird beim Starten eines PCs aktiv: sie „kann“ dhcp und führt die Abfragen im Netz durch.

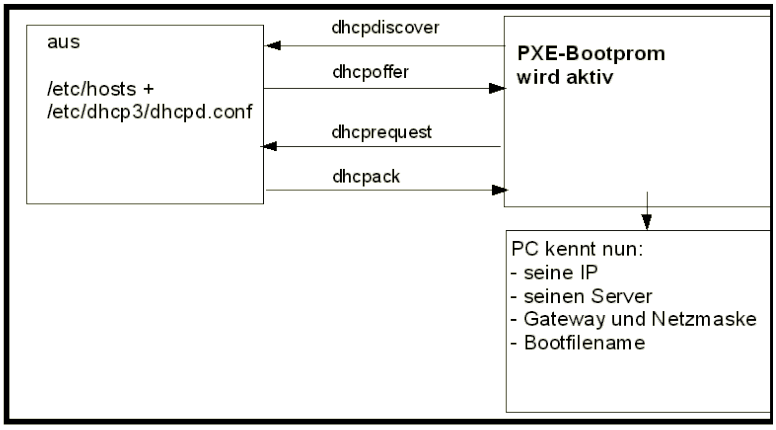


Abbildung 54: Schritt 1 beim PXE-Boot

Der PC kennt zu Beginn lediglich seine Hardware-Adresse (= hardware ethernet, MACNummer der Netzwerkkarte), bestehend aus sechs zweistelligen Hexadezimalzeichen.

Die Firmware schickt damit eine Rundfrage ins Netz. Es ist eine *DHCPDISCOVER*Anfrage über Standard-Port per Broadcast (= an alle Rechner im Netz): „Ich brauche eine IP-Nummer und wer ist mein dhcp-Server?“ (Discover= entdecken)

Mittels **DHCPOFFER** macht der dhcp-Server diesbezüglich einen Vorschlag. (offer=anbieten)

DHCPREQUEST ist die Antwort des Clients an den Server (wenn er die angebotene IP akzeptiert; Hintergrund ist hier: Es können in einem Netz mehrere dhcp-Server tätig sein.). Der Client fordert damit die angebotene Adresse an. (request=Anfrage)

Mit **DHCPACK** bestätigt der dhcp-Server diese Anforderung des Clients. Die Informationen werden an den Client übertragen. (acknowledge=bestätigen)

Am Bildschirm des bootenden PCs können diese Prozesse mitverfolgt werden. Nach den ersten Systeminformationen meldet sich das PXE-BOOTPROM mit seinen technischen Daten und stellt seine „CLIENT MAC ADDR“ dar. Im Anschluss zeigt ein sich drehendes Pipe-Zeichen die Dauer der Anfrage des Clients an. Wird das bewegliche Zeichen durch einen Backslash ersetzt, hat der dhcp-Server ein Angebot gemacht („CLIENT IP, MASK, DHCP IP, GATEWAY IP“).
Kurze Zeit später – wenn alles funktioniert hat – meldet das PXE: „My IP ADDRESS SEEMS TO BE . . . "

Nach dem Empfang und der Verarbeitung dieser Konfigurationsinformationen durch den PC ist dieser als Netzwerkteilnehmer ordentlich konfiguriert. Der nächste Schritt ist, das in den Konfigurationsinformationen angegebene Bootfile (bootimage) zu laden.

9.2.4 pxelinux wird geladen

Das bootimage wird per tftp (trivial file transfer protocol) geladen. (Meldung auf dem PC-Bildschirm: „LOADING“). Das Protokoll tftp ist zum Übertragen von Dateien, bei dem sich der Client nicht authentifizieren muss. Das heißt, die über tftp ladbaren Dateien sind für alle im Netz verfügbar. Daher wird der Zugriff per tftp auf ein bestimmtes Verzeichnis (mit Unterverzeichnissen) beschränkt. Gewöhnlich ist dieses Verzeichnis /tftpboot. Konfiguriert ist dies in

der Konfigurationsdatei des inetd (/etc/inetd.conf), der den eigentlichen tftpd bei Bedarf startet. (z.B. `tftpd -p -u tftp -s /tftpboot`).

Der Ladevorgang gemäß dem PXE-Standard ist dabei mehrstufig:
In der ersten Stufe wird die per tftp übermittelte Datei (üblicherweise /tftpboot/linux/pxelinux.0) geladen und gestartet.
Das Programm pxelinux.0 sucht bei Ausführung im Verzeichnis /tftpboot/linux/pxelinux.cfg nach Konfigurations- bzw. Bootinformationen. Dabei wird zunächst nach PC-spezifischen Informationen gesucht. Eine solche PC-spezifische Datei basiert auf der Hardwareadresse (MAC-Adresse) der Netzwerkkarte im Dateinamen. Die Datei ist eine *Einweg*-Datei (named pipe) und kann daher nur einmal gelesen werden. Der Hardwareadresse im Dateinamen werden dabei immer die zwei Ziffern 01 vorangestellt. Alle Zeichenpaare werden durch ein Minuszeichen verknüpft, z.B. 01-00-0c-29-11-6b-d2 für eine Netzwerkkarte mit MAC: 00:0C:29:11:6B:D2. Wird eine solche Datei nicht gefunden wird nach einer Datei gesucht deren Namen der Hexadezimaldarstellung der IP-Adresse entspricht. Ist auch keine solche PCspezifische Datei vorhanden, wird pxelinux.0 den Dateinamen (von hinten beginnend) immer weiter verkürzt suchen, bis die Suche ergebnislos verlaufen ist und bei der Datei „default“ endet. Diese Datei enthält den Befehl hdboot. Lädt der PC diese Datei, findet also keine Installation statt, sondern das lokal installierte Betriebssystem wird gestartet.

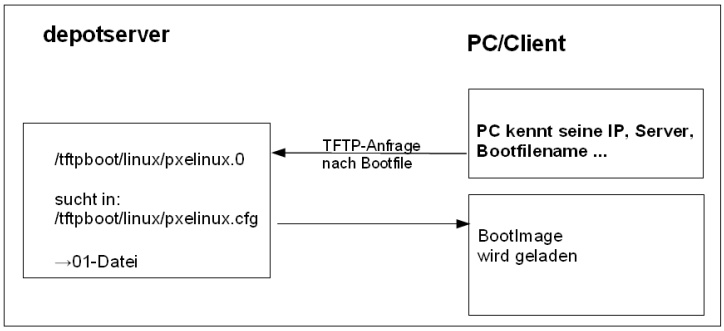


Abbildung 55: Schritt 2 beim PXE-Boot

Um für einen bestimmten PC eine Reinstallation einzuleiten, wird das Programm pxelinux.0 dazu gebracht, in einer zweiten Stufe ein Installationsbootimage zu laden. Dazu wird mit Hilfe des *opsipxeconfd* eine PC-spezifische Datei in `/tftpboot/linux/pxelinux.cfg` erzeugt, in der unter anderem der Befehl zum Laden des eigentlichen Installationsbootimages liegt. Weiterhin findet sich hier der PC-spezifische Schlüssel zur Entschlüsselung des pcpatch-Passwortes. Diese Datei wird als *named pipe* erzeugt und ist damit eine *Einweg*-Datei die durch einmaliges Lesen von selbst verschwindet. Details hierzu in den Kapiteln zur Absicherung der Shares und zum *opsipxeconfd*.
Linux Installationsbootimage wird geladen
Basierend auf den Informationen die das pxelinux.0 aus der named pipe gelesen hat, wird nun per tftp vom *opsi-server* das eigentliche Installationsbootimage geladen. Dieses besteht üblicherweise aus dem Kernel (`/tftpboot/linux/install`) in dem dazugehörigen "initrd" (initiale root disc) Filesystem (`/tftpboot/linux/miniroot.gz`).
Das Bootimage, das nun geladen wird, ist Linux basiert und hat etwa eine Größe von 65 MB.

9.2.5 PC-Client bootet von CD

Analog zu dem Bootvorgang per tftp mit Hilfe des PXE-bootproms kann das Installationsbootimage auch direkt von der opsi-bootcd geladen werden.

Diese Möglichkeit bietet sich bei folgenden Voraussetzungen an:

- der Client verfügt über kein PXE;

- es gibt kein dhcp;
- es gibt dhcp aber es sollen dort keine Einträge zu den Clients gemacht werden und die Hardwareadressen der Clients sind nicht bekannt;
- es gibt dhcp aber dieses ist nicht korrekt konfigurierbar

Entsprechend der unterschiedlichen Situationen müssen dem Bootimage auf der CD unterschiedlich viele Informationen interaktiv bereitgestellt werden. Im einfachsten Fall müssen überhaupt keine Angaben gemacht werden.

Lesen Sie hierzu auch das Kapitel *Anlegen eines neuen opsi-Clients mit Hilfe der opsi-client-bootcd* in Getting-Started Handbuch.

9.2.6 Das Linux Installationsbootimage bereitet die Reinstallation vor

Das Bootimage startet eine erneute dhcp-Anfrage und konfiguriert sich entsprechend sein Netzwerkinterface. Danach werden über den *opsi-webservice* die Konfigurationsdaten für diesen Client geladen.

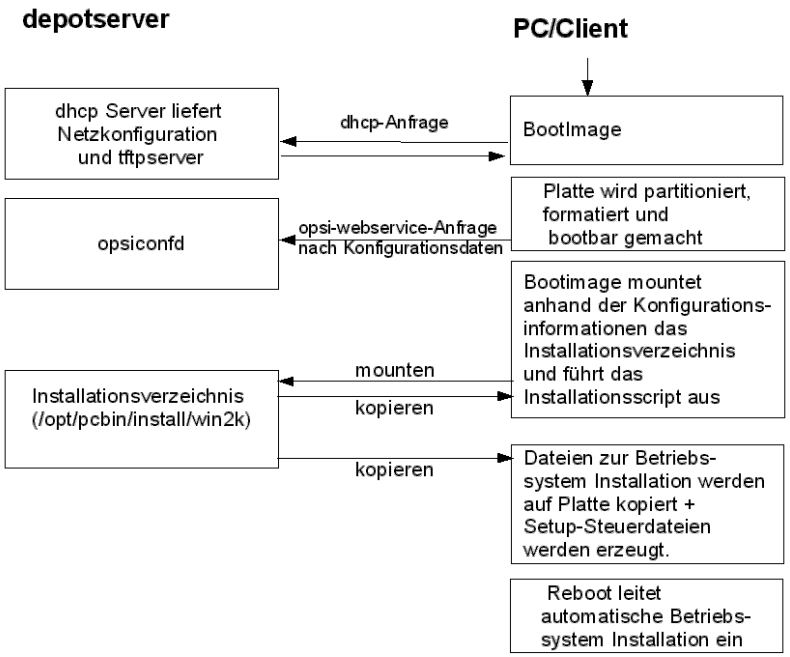


Abbildung 56: Ueber PXE-Boot geladenes Bootimage bereitet Festplatte zur Betriebssysteminstallation vor

Ergänzt wird dieses Informationspaket durch Angaben aus der dhcp-Antwort (z.B. wer ist der tftp-Server) sowie mit über den Webservice ermittelte Informationen. Die gesammelten Informationen werden für die Weiterverarbeitung durch das eigentliche Installationsskript bereitgestellt.

Nun wird das Passwort des Installations-Users pcpatch mit Hilfe des übergebenen Schlüssels entschlüsselt und der angegebene Installationsshare gemountet. Jetzt kann das auf dem gemounteten Share liegende Installationsskript für das zu installierende Betriebssystem gestartet werden. Die Abläufe in diesem Skript sind abhängig von dem zu installierenden Betriebssystem. Im Folgenden werden beispielhaft die Abläufe für eine Windows-XP-Installation skizziert.

Vorbereitung der Festplatte: Auf der Platte wird eine 6 GB große FAT32 Partition angelegt, formatiert und bootfähig gemacht.

Kopieren der Installationsdateien: Die Dateien für die Installation des Betriebssystems werden von dem Installationsshare des Servers (z.B. /opt/pcbin/install/winxppro/i386) auf die lokale Platte kopiert. Das Gleiche gilt für das Setup-Programm des 'opsi-client-agent's zur Einrichtung der automatischen Softwareverteilung auf dem PC.

Einpflegen der Konfigurationsinformationen: Unter den auf die lokale Platte kopierten Dateien finden sich auch Konfigurations- und Steuerdateien, die Platzhalter enthalten. Durch ein spezielles Skript (patcha) werden diese durch entsprechende Parameter aus dem Informationspaket ersetzt (gepatcht), welches das Bootimage zuvor aus Konfigurationsdateien und dhcp-Antwort bereitgestellt hat. Ein Beispiel für eine zu patchende Datei ist die unattend.txt. Sie steuert das „unbeaufsichtigte“ Installieren von Windows 2003/XP.

Reboot vorbereiten: Der *Bootloader* wird so konfiguriert, dass beim nächsten Boot der Rechner via *ntloader* in das Windows Setup-Programm startet. Der Aufruf ist dabei mit der Option versehen, die gepatchte unattend.txt als Steuerdatei zu verwenden.

Reboot: Da in `/tftpboot/linux/pxelinux.cfg` nun keine PC-spezifische Datei mehr vorhanden ist, wird in Stufe 1 des PXE-Boots der Befehl hdboot aus der Datei default geladen. Damit wird der lokale Bootloader gestartet und die Betriebssysteminstallation gestartet.

Die beschriebenen Abläufe werden von dem für diese Installation angegebenen Python-Script gesteuert.

9.2.7 Die Installation von Betriebssystem und opsi-client-agent

Die Installation des Betriebssystems ist ein unattended Setup wie es von Microsoft vorgesehen ist. Dabei werden die Standardmöglichkeiten der Hardwareerkennung genutzt. Im Gegensatz zu einer normalen Installation von CD können auf der Installations-Partition schon aktualisierte Treiber und Servicepacks eingepflegt werden, damit diese schon direkt bei der Erstinstallation verwendet werden.

Zu einer unattended Installation gehört die Möglichkeit, nach Abschluss der eigentlichen Betriebssysteminstallation automatisch noch weitere Installationen starten zu können. Dieser Mechanismus wird genutzt, um das Setup des 'opsi-client-agent's auszuführen und damit die automatische Softwareverteilung einzubinden. In der Registry wird eingetragen, dass sich der Rechner immer noch im Reinstallationsmodus befindet.

Nach dem abschließenden Reboot starten nun vor einem Login die opsi-Programme zur Softwareverteilung. Diese Software erkennt anhand der Registry den Reinstallationsmodus. Dieser Modus hat hier zur Folge, dass alle Softwarepakete, für welche der Installationsstatus *installed* oder die angeforderte Aktion *setup/update* ist, nun installiert werden. Auf diese Weise werden sämtlich Pakete, die vor der Reinstallation des Betriebssystems auf diesem PC waren, automatisch wieder eingespielt. Erst nach Abschluss aller Installationen wird der Reinstallationsmodus zum Standard-Bootmodus zurückgeschaltet. (Im Gegensatz zum Reinstallationsmodus, werden im Standard-Bootmodus nur Pakete installiert, bei denen die Angeforderte Aktion *setup/update* ist.) Damit ist der PC fertig installiert.

9.2.8 Funktionsweise des patcha Programms

Wie oben erläutert werden vom Bootimage (genauer gesagt vom Programm `/usr/local/bin/master.py`) die Konfigurationsinformationen aus dem *opsi-webservice* und dhcp gesammelt, um sie dann in entsprechende andere Konfigurationsdateien wie z.B. die `unattended.txt` einzupflegen. Das Einpflegen übernimmt das Programm `/usr/local/bin/patcha`.

Das Skript gleicht anhand eines Suchmusters *@flagname()* *eine Konfigurationsdatei mit den Einträgen aus einer anderen Datei (hier cmdline) ab, die Einträge der Art "Flagname=Wert" enthalten muss und patcht diese bei Übereinstimmung des Suchmusters. Das Suchmuster kann nach dem Flagnamen einen ""* enthalten und muß einen oder beliebig viele "#" als Abschluß enthalten. Default wird `/proc/cmdline benutzt`. Wenn man `patcha` ohne irgendwelche Optionen und ohne Dateiübergabe aufruft, werden die "Flagname=Wert"-Paare aus der `/proc/cmdline` ausgegeben.

Wenn man `patcha <dateiname>` eingibt, patcht er die Datei mittels der `/proc/cmdline`.

Eine andere cmdline als /proc/cmdline, gibt man mit `patcha -f <andere_cmdline>` mit. Ohne zusätzlich mitgegebenen Dateinamen werden die Werte der andere_cmdline ausgegeben, mit Dateiname wird die Datei mit den Werten aus andere_cmdline gepatcht.

```
Usage: patcha [-h|-v] [-f <params file>] <patch file>

Fill placeholders in file <patch file>
Options:
-v Show version information and exit
-h Show this help
-f <params file> File containig key value pairs
If option not given key value pairs from kernel cmdline are used
```

`patcha` patcht nur einen Tag pro Zeile.

Der Platzhalter wird auf die Länge des zu ersetzenden Wertes getrimmt bzw erweitert und dann ersetzt. D.h unabhängig von der Länge des Platzhalters wird dieser durch den Wert ersetzt. Anhängende Zeichen bleiben anhängend. Beispiel:

Mit der Datei

```
cat try.in
tag1=hallohallohallo1 tag2=t2
```

und der Datei

```
cat patch.me
<#@tag1############################>
<#@tag2############################>
<#@tag1#>
<#@tag2#>
<#@tag1*############################>
<#@tag2*############################>
<#@tag1*#>
<#@tag2*#>
<#@tag1#><#@tag1#####>
<#@tag2*#######><#@tag1#>
```

ergibt

```
./patcha -f try.in patch.me
cat patch.me
<hallohallohallo1>
<t2>
<hallohallohallo1>
<t2>
<hallohallohallo1>
<t2>
<hallohallohallo1>
<t2>
<hallohallohallo1><#@tag1#####>
<t2><#@tag1#>
```

9.2.9 Aufbau der Produkte zur unattended Installation

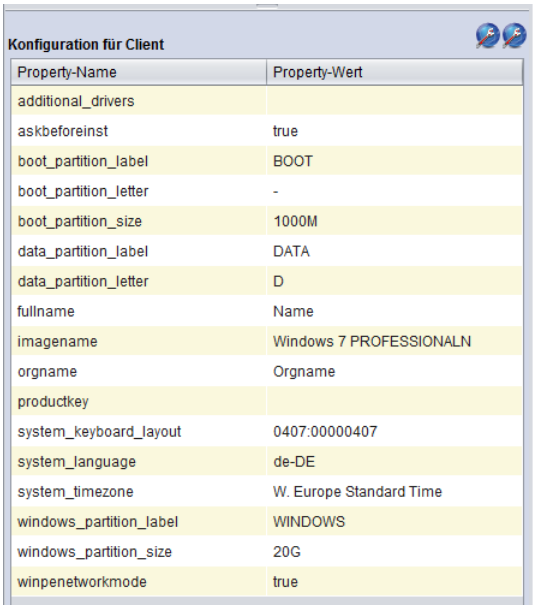
Die Informationen zum *Aufbau der Produkte zur unattended Installation* finden Sie im Handbuch opsi-getting-started.

9.2.10 Vereinfachte Treiberintegration in die automatische Windowsinstallation

Die Informationen zum *Vereinfachte Treiberintegration in die automatische Windowsinstallation* finden Sie im Handbuch opsi-getting-started.

9.3 Hinweise zu den NT6 Netbootprodukten (Vista / Win7 / 2008)

Die Netbootprodukte zur Installation von NT6 Betriebssystemen enthalten eine Fülle von Produktproperties, welche in Ihrer Funktion in der Folge erläutert werden sollen:



Konfiguration für Client

Property-Name	Property-Wert
additional_drivers	
askbeforeinst	true
boot_partition_label	BOOT
boot_partition_letter	-
boot_partition_size	1000M
data_partition_label	DATA
data_partition_letter	D
fullname	Name
imagename	Windows 7 PROFESSIONALN
orgname	Orgname
productkey	
system_keyboard_layout	0407:00000407
system_language	de-DE
system_timezone	W. Europe Standard Time
windows_partition_label	WINDOWS
windows_partition_size	20G
winpenetworkmode	true

Abbildung 57: NT6 Productproperties

additional_drivers
: Liste von Verzeichnissen unterhalb von `<productid>\drivers\drivers\additional`. Alle Treiberverzeichnisse unterhalb der angegebenen Verzeichnisse werden unabhängig von der automatischen Treibererkennung zusätzlich in die Installation mit eingebunden. Wir hierüber Treiber für ein Gerät eingebunden, so wird für dieses Gerät kein weiterer Treiber über die automatische Treiberintegration mehr eingebunden.

askbeforeinst
: Soll vor Beginn der Installation gefragt werden.

boot_partition_label
: Label der *boot_partition* (Bitlocker Partion)

boot_partition_letter
: Laufwerksbuchstabe der *boot_partition* (Bitlocker Partion)

boot_partition_size
: Größe der *boot_partition* (Bitlocker Partion). 0 = keine Erstellen

data_partition_label
: Label der Datenpartition, wenn eine erstellt wird.

data_partition_letter
: Laufwerksbuchstabe der Datenpartition, wenn eine erstellt wird.

fullname
: Vollständiger Name des Lizenznehmers wie er der Installation übergeben wird.

imagename
: Name der Variante des Betriebssystems das zu installieren ist.

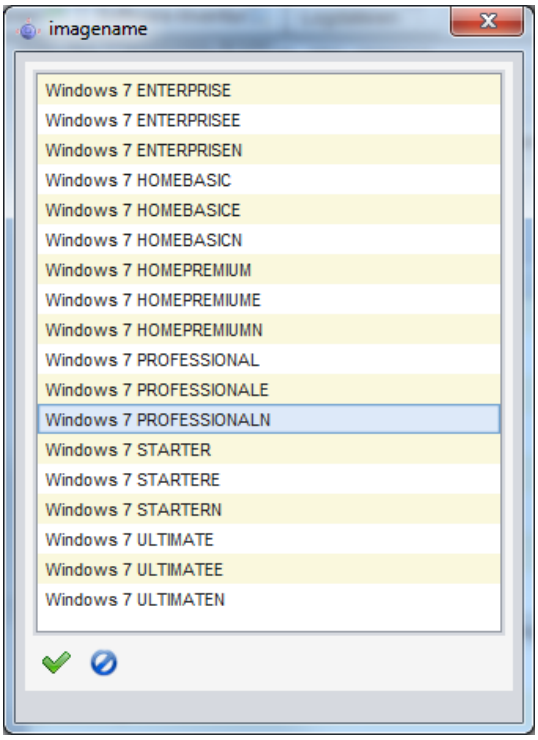


Abbildung 58: NT6 Imagenames

orgname
Vollständiger Name der Firma / Organisation des Lizenznehmers wie er der Installation übergeben wird.

productkey
Lizenzschlüssel zur Installation. Wird nur ausgewertet wenn der Hostparameter `license-management.use` auf *false* steht. Ansonsten wird der Lizenzschlüssel aus dem Lizenzmanagement geholt.

system_keyboard_layout
Sprachauswahl für die Tastatur.

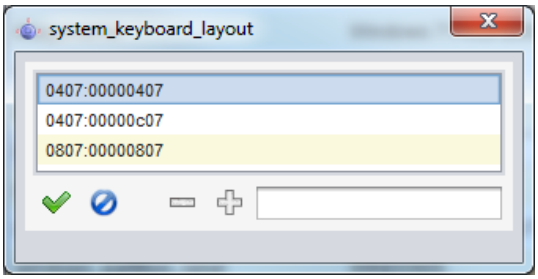


Abbildung 59: Sprachauswahl für die Tastatur

system_language
Sprachauswahl für das System.

system_timezone
Zeitzoneneinstellung

windows_partition_label
Label der Partition (Festplatte C:) auf die das Betriebssystem installiert werden soll.

windows_partition_size
Größe der Partition (Festplatte C:) auf die das Betriebssystem installiert werden soll. Die Angabe kann in Prozent der Festplatte oder in absoluten Zahlen (G=Gigabyte) erfolgen. Wird ein anderer Wert als 100% gewählt, so wird auf dem verbleibenden Rest der Fasplatte eine *data_partition* angelegt.

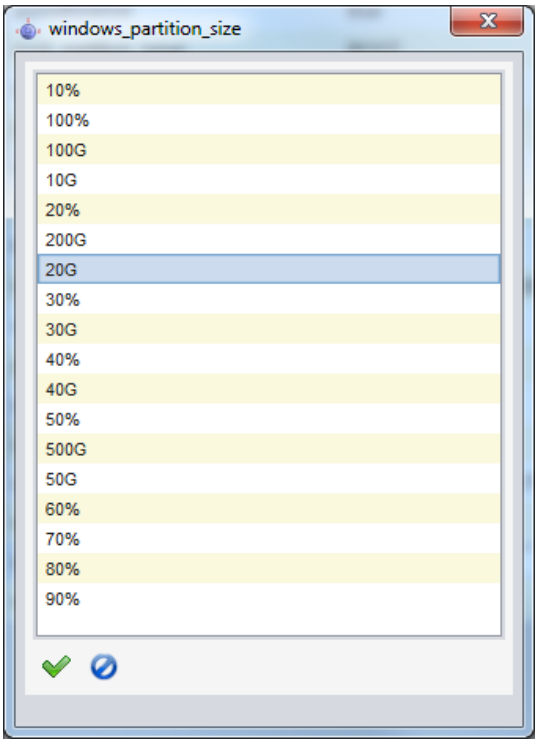


Abbildung 60: Größe der C: Partition

winpenetworkmode
Soll die Betriebssysteminstallation über den gemounteten Netzwerkshare vom PE aus erfolgen (true) oder sollen alle Installationsdateien vorher auf die Festplatte kopiert werden (false).

9.4 Ntfs-images (write + restore)

Mit den Produkten *ntfs-write-image* und *ntfs-restore-image* können Sie Abbilder von Partitionen sichern bzw. wiederherstellen. Ziel bzw. Quelle der Imagedatei müssen auf dem *opsi-depotserver* liegen und werden per ssh (user pcpatch) erreicht und im Produktproperty angegeben.

Entsprechende Produkte zum Sichern und Wiederherstellen von NTFS-Partitionen gibt es auch auf der opsi-clientbootcd und sind in einem gesonderten Manual beschrieben.

9.5 memtest

Das Produkt *memtest* dient dazu einen Memory-Test des Clients durchzuführen.

9.6 hwinvent

Das Produkt *hwinvent* dient dazu eine Hardwareinventariserung des Clients durchzuführen.

9.7 wipedisk

Das Produkt *wipedisk* überschreibt die gesamte Festplatte (partion=0) oder einzelne Partitionen mit unterschiedlichen Mustern. Die Anzahl der Schreibvorgänge wird über das Product-Property *iterations* gesteuert (1-25).

10 Inventarisierung

Zur Inventarisierung stehen die Localbootprodukte `hwaudit` und `swaudit` sowie das Netboot Produkt `hwinvent` zur Verfügung.

10.1 Hardware Inventarisierung

Die Hardwareinventarisierung ist unter opsi über eine Konfigurationsdatei gesteuert. Das bedeutet, das die Information wie und welche Daten erhoben werden, nicht in den entsprechenden Produkten `hwaudit` und `hwinvent` fest verdrahtet sind. Vielmehr werden diese Produkte über eine Konfigurationsdatei gesteuert. Dazu wird die Konfigurationsdatei bei jeder Ausführung über den opsi Webservice eingelesen und interpretiert. Gleichzeitig steuert diese Konfigurationsdatei auch den Aufbau der Datenbank, so dass eine Erweiterung dieser Konfigurationsdatei auch eine Erweiterung der Datenhaltung nach sich zieht.

Die Konfigurationsdatei ist die `/etc/opsi/hwaudit/opsihwaudit.conf`.
In dieser Datei werden alle zu Inventarisierenden Objekte definiert und beschreiben, wie die zu diesem Objekt gehörenden Daten zu erheben sind (unter Linux und unter Windows). Gleichzeitig wird darüber auch die dazu gehörige Datenstruktur definiert. Zur Vereinfachung enthält diese Konfigurationsdatei Vererbungsmechanismen die an eine Objektorientierung angelehnt sind. Hintergrund hierfür ist die Tatsache, dass viele Objekte identische Datenfelder wie z.B. `Name` und `Vendor` enthalten. Diese allgemeinen Informationen werden so in *virtual* Hardwareklassen definiert. Die eigentlichen Inventarisierungsobjekte sind dann *structural* Hardwareklassen, welche viele Eigenschaften von übergeordneten *virtual* Klassen erben können.

Zur Erläuterung dieses Mechanismus ein Beispiel:
So definiert die Knfigurationsdatei zunächste eine *virtual Class* Namens *"BASIC_INFO"*. Diese definiert die Eigenschaften (*Values*):

- "name"
- "description"

Als nächstes folgt die *virtual Class* Namens *"HARDWARE_DEVICE"* welche alle Eigenschaften von *"BASIC_INFO"* erbt und folgende zusätzliche definiert:

- "vendor"
- "model"
- "serialNumber"

Als nächstes folgt als erstes Objekt welche wir in der Inventarisierung auch finden, die erste *structural Class* Namens *"COMPUTER_SYSTEM"*, welche alle Eigenschaften von *"HARDWARE_DEVICE"* erbt und folgende zusätzliche definiert bzw. überschreibt:

- "name"
- "systemType"
- "totalPhysicalMemory"

Im Rahmen der Definition einer Klasse und ihrer *Values* werden verschiedene Eigenschaften beschrieben:

- Klassen definition:
  - "Type"
    ist "STRUCTURAL" oder "VIRTUAL"
  - "Super"
    gibt die Klasse an von der geerbt wird.
  - "Opsi"
    gibt den Namen der Klasse an, der auch später in opsi als Anzeigenamen verwendet wird.

Weiterhin können in der Klassendefinition angegeben werden, wie diese Daten erhoben werden. Diese Informationen können aber auch bei der Definition der *Values* stehen.

- Für die Inventarisierung unter Linux:
  - "Linux": "[<command>]<parameter>"
    Ausführung des Kommandozeilenprogramms <command> mit dem Argument <parameter>.
  - "Python": "<python code with place holder>"
    Ausführung des angegeben Python codes wobei zunächst der Platzhalte durch die schon ermittelten Werte ersetzt wird.
- Für die Inventarisierung unter Windows:
  - "WMI": "<wmi select statement>"
    auszuführende WMI Abfrage
  - "Cmd": "<Python text object with place holder>"
    Der Platzhalter ist in diesem Fall der relative Pfad zu einem ausführbarem Programm, dessen Ausgabe den Platzhalter ersetzt.
  - "Registry": "[<registry key>] <value name>"
    Aus der Registry wird in <registry key> der Wert von <value name> ausgelesen.
    Das Auslesen der Registry erfolgt Architektur spezifisch. Das heißt, auf einem 64 Bit System wird der 64 Bit Zweig der Registry ausgelesen.
- Valuedefinition:
  - "Type": "<MySQL Datenbanktyp>"
    <MySQL Datenbanktyp> gibt den Datentyp an in dem dieser Wert in der Datenbank angelegt wird.
  - "Scope": "<scope>"
    das Feld <scope> wird folgendsermaßen verwendet:
    "g" bedeutet: Dieses Attribut ist bei allen Geräten dieses Typs gleich.
    "i" bedeutet: Dieses Attribut kann bei Geräten dieses Typs unterschiedliche Werte haben.
  - "Opsi": "<id>"
    dabei ist <id> der opsi interne Name des Feldes. Dieser kann zur Ausgabe über die Dateien in `/etc/opsi/hwaudit/locales` wiederum lokalisiert werden.
  - "WMI": "<id or command>"
    dabei ist <id or command> entweder der Name unter dem der in der Klassen definition angegebene WMI Befehl den Wert ausgibt oder ein eigener WMI Befehl.
  - "Linux": "<id>"
    dabei ist <id> der Name unter dem der in der Klassen definition angegebene Linux Befehl den Wert ausgibt.
  - "Condition": "<condition>"
    dabei ist <condition> eine Bedingung die erfüllt sein muss, damit der *Value* ermittelt wird. So legt z.B. die <condition> "vendor=[dD]ell*" fest, das der schon erhobene Wert von "vendor" *Dell* oder *dell* enthalten muss.

Hierzu als Beispiel die Klasse "COMPUTER_SYSTEM":

```
{
    "Class": {
        "Type":    "STRUCTURAL",
        "Super":   [ "HARDWARE_DEVICE" ],
        "Opsi":    "COMPUTER_SYSTEM",
        "WMI":     "select * from Win32_ComputerSystem",
        "Linux":   "[lshw]system"
    },
    "Values": [
        {
            "Type":    "varchar(100)",
            "Scope":   "i",
            "Opsi":    "name",
            "WMI":     "Name",
            "Linux":   "id"
        },
        {
            "Type":    "varchar(50)",
            "Scope":   "i",
            "Opsi":    "systemType",
            "WMI":     "SystemType",
            "Linux":   "configuration/chassis"
        },
        {
            "Type":    "bigint",
            "Scope":   "i",
            "Opsi":    "totalPhysicalMemory",
            "WMI":     "TotalPhysicalMemory",
            "Linux":   "core/memory/size",
            "Unit":    "Byte"
        },
        {
            "Type":    "varchar(50)",
            "Scope":   "i",
            "Opsi":    "dellexpresscode",
            "Condition": "vendor=[dD]ell*",
            "Cmd": "#dellexpresscode\dellexpresscode.exe#.split('=')[1]",
            "Python":   "str(int(#{'COMPUTER_SYSTEM':'serialNumber','CHASSIS':'serialNumber'}#
        }
    ]
},
```

Besonders interessant ist hier der letzte Value "dellexpresscode":
Dieser ist nur sinnvoll, wenn es sich auch um einen Dell-Rechner handelt, daher die Condition.
Unter Windows wird das Kommandozeilen Programm `dellexpresscode.exe` ausgeführt, welches sich von der `hwaudit.exe` aus gesehen im Unterverzeichnis `dellexpresscode\` befindet. Diese produziert eine Ausgabe in der Form: *dellexpresscode=123456789*. Durch den hinter dem Platzhalter befindlichen `.split('=')[1]` wird der Wert hinter dem Gleichheitszeichen verwendet.
Unter Linux wird geprüft in welchem Element (*COMPUTER_SYSTEM* oder *CHASSIS*) bei *serialNumber* ein Wert gefunden wurde, und dieser dann zur Berechung des Dell expresscodes verwendet.

Die Opsi-Namen der Values werden über die Dateien `/etc/opsi/hwaudit/locales/*` übersetzt. Bsp. `/etc/opsi/hwaudit/locales/de_DE`:

```
COMPUTER_SYSTEM = Computer
COMPUTER_SYSTEM.systemType = Typ
```

Der Klassenname COMPUTER_SYSTEM wird übersetzt in "Computer". Das Opsi-Attribut "systemType" der Klasse COMPUTER_SYSTEM wird übersetzt in "Typ". Abschliessend noch der Hinweis: Wenn ein neues Feld erzeugt wird, sollte man dieses in den locale-Dateien anlegen, auch wenn man den Begriff selber nicht übersetzt. Dadurch wird vermieden, dass bei der Laufzeit "*Warning*" Meldungen produziert werden.

10.2 Software Inventarisierung

Die Softwareinventarisierung findet über das Localbootprodukt `swaudit` statt. Dabei werden die Informationen aus dem Uninstallzweig der Registry erhoben und durch zusätzliche Informationen zu Hotfixes und Lizenzkeys ergänzt.

Der Quellcode diese Paketes wir hier verwaltet:
https://svn.opsi.org/listing.php?repname=swaudit

11 opsi-server

11.1 Überblick

Die Funktionalitäten eines *opsi-servers* lassen sich auf vielen Business gängigen Linuxdistributionen installieren:

Grob lassen sich zwei wichtige Funktionalitäten unterscheiden, die auf einem Server vereint sein können:

- *opsi-configserver*
  Die Funktionalität des configserver umfasst die Speicherung und Verarbeitung der Konfigurationsdaten in unterschiedlichen Backends und deren Bereitstellung über einen Webservice und auf Kommandozeile.
- *opsi-depotserver*
  Die Funktionalität des depotserver umfasst die Speicherung der eigentlichen Installationsdateien der zu verteilenden Software, Betriebssysteme, Bootimages und deren Bereitstellung für den Client per smb, cifs/https, tftp.

Die aus diesen Diensten entstehenden Hardwareanforderungen sind in der Regel gering, so das ein Betrieb eines opsi-servers in einer Virtualisierungsumgebung kein Problem darstellt.

11.2 Installation und Inbetriebnahme

Die Installation und Inbetriebnahme eines *opsi-servers* ist in dem gesonderten Handbuch: *opsi-getting-started* ausführlich erläutert.

11.3 Samba Konfiguration

Um den Client-PCs Zugriff auf die Softwarepakete zu ermöglichen, stellt der opsi-server Shares bereit, die von den Clients als Netzlaufwerke gemountet werden können. Für die Windows-Clients wird dazu die Software SAMBA eingesetzt. Die Korrekte Sambakonfiguration können Sie erstellen bzw. reparieren durch den Aufruf von:

```
opsi-setup --auto-configure-samba
```

Nach einer Änderung der Samba-Konfigurationsdateien ist ein reload der Samba-Software notwendig (`/etc/init.d/samba reload`).

11.4 Der Daemon opsiconfd

Der *opsiconfd* ist der Zentrale Konfigurations-Daemon von opsi. Alle Client-Komponenten (opsi-client-agent, opsi-configed, opsi-linux-bootimage, ...) verbinden sich mit diesem Service um auf die Konfigurationen in den Backends zuzugreifen. Der *opsiconfd* wird über die Datei `/etc/opsi/opsiconfd.conf` konfiguriert. Die einzelnen Konfigurations-Optionen sind in dieser Datei dokumentiert.

- `[global] admin networks`:
  Über diese Option kann der administrative Zugriff auf den *opsiconfd* auf Verbindungen von bestimmten Netzwerkadressen eingeschränkt werden.
  Es können mehrere Netzwerkadressen durch Kommas getrennt angegeben werden.
  Nicht-administrative Client-Verbindungen können auch aus anderen Netzwerken erfolgen.

11.5 Notwendige System-User und Gruppen

- User *opsiconfd*
  Dies ist der user unter dem der opsiconfd Deamon läuft.
- User *pcpatch*
  Dies ist der user, den der *opsi-client-agent* verwendet um den *depotshare* zu mounten und von diesem zu lesen. Dieser System-User pcpatch hat üblicherweise die user-ID 992. Dieser User hat per Voreinstellung das Heimatverzeichnis `/var/lib/opsi`.Setzen und Ändern Sie das Passwort mit `opsi-admin -d task setPcpatchPassword`.
- Gruppe *pcpatch*
  Neben dem User pcpatch gibt es noch die Gruppe pcpatch. Die meisten Dateien sind sowohl für den User als auch für die Gruppe im Vollzugriff. Die Systemadministratoren des opsi-servers sollten daher Mitglieder der Gruppe pcpatch sein.
- Gruppe *opsiadmin* Die Mitglieder dieser Gruppe können sich gegenüber dem opsi-webservice Authentifizieren und damit z.B. mit dem opsi-configed arbeiten. Daher sollten alle Mitarbeiter die mit opsi arbeiten, Mitglied dieser Gruppe sein.

11.6 Notwendige Shares

- Bereich: *Depotshare* mit Softwarepaketen (*opt_pcbin*)
  Auf dem depot-Share liegen die für die Installation durch das Programm opsi Winst vorbereiteten Softwarepakete. In der Voreinstellung liegt dieses Verzeichniss auf dem opsi-server unter `/opt/pcbin`. Unterhalb von diesem Verzeichnis findet sich das Verzeichnis `install` und in diesem für jedes Softwarepaket ein Verzeichnis mit dem Namen des Softwarepakets. Wiederum unterhalb dieses Verzeichnisses liegen dann die Installationsskripte und -dateien.

Tipp
Das Verzeichnis `/opt/pcbin/install` wird in Zukunft durch das Verzeichnis `/var/lib/opsi/depot` ersetzt. Das Verzeichnis wird mit dem Freigabe-Namen *opsi_depot* per Samba read-only exportiert.

- Bereich: Arbeitsverzeichnis zum Pakethandling (*opsi_workbench*)
  Unter `/home/opsiproducts` ist der Bereich um Pakete zu erstellen und in dem Pakete vor der Installation mit opsi-package-manager abgelegt werden sollen. Dieses Verzeichnis ist als share *opsi_workbench* freigegeben.
- Bereich: Konfigurationsdateien File-Backend (*opsi_config*)
  Unter `/var/lib/opsi` liegen die Konfigurationsdateien des file Backends. Dieses Verzeichnis ist als share *opsi_config* freigegeben.
  CAUTION: Wenn Sie über diesen Share auf den Dateien arbeiten, verwenden Sie keine Editoren die das Dateiformat (Unix/DOS) verändern und entfernen Sie Sicherungsdateien wie z.B. *.bak.

11.7 Problem-Management

Sollten Probleme im Betrieb mit dem opsi-Server auftreten so sollten folgende Dinge überprüft und ausgeführt werden. Die Erfahrung zeigt, dass die Kombination der folgender Befehle, die meisten Probleme behebt.

- Erreichbarkeit und Auslastung vom *opsi-webservice* prüfen:
  https://<server-ip>:4447/info per Browser aufrufen. Wenn dies schon nicht geht, weiter mit nächstem Schritt. Wenn diese Seite angezeigt wird: Prüfen ob die Auslastung vom *opsi-webservice* zu hoch ist. Am besten auch den freien Speicherplatz auf dem Server überprüfen (Weiter unten auf der info-Seite).
- Prüfen ob die benötigten Daemons laufen, eventuell neustarten:

```
ps -ef | grep opsiconfd
ps -ef | grep opsipxeconfd
/etc/init.d/opsiconfd restart
/etc/init.d/opsipxeconfd restart
```

- Konfiguration neu einlesen:

```
opsi-setup --init-current-config
```

- Rechte auf die opsi-relevanten Dateien und Ordner neu setzen:

```
opsi-setup --set-rights
```

- Backend aufräumen:

```
opsi-setup --cleanup-backend
```

- Prüfen ob Samba läuft, eventuell neustarten:

```
ps -ef | grep mbd
```

Es muss mindestens ein nmbd und mindestens ein smbd prozess laufen. Eventuell samba neustarten:

```
/etc/init.d/samba restart
```

oder

```
service nmbd restart
service smbd restart
```

- Neusetzen des pcpatch-Passworts:

```
opsi-admin -d task setPcpatchPassword
```

12 Security

12.1 Einführung

Opsi ist ein mächtiges Werkzeug zur zentralen Administration vieler Clients. Damit steht der *opsi-server* natürlich auch im besonderen Fokus der Sicherheitsbetrachtung. Wer den *opsi-server* kontrolliert, kontrolliert auch die Clients. Wer einem Client davon überzeugen kann er sei der richtige *opsi-server*, der kontrolliert den Client.

Wieviel Energie und Geld Sie in die Absicherung der opsi Infrastruktur stecken sollten hängt von Ihren Sicherheitsbedürfnis und vom Einsatzszenario ab. Ein *opsi-server* in der *cloud* ist z.B. gefährdeter als einer in einem abgeschlossenen Inselnetz.

Im folgenden haben wir die wichtigsten uns bekannten Maßnahmen und Probleme zusammengetragen.

Wir danken an dieser Stelle allen Kunden und Anwendern die uns auf Probleme hingewiesen und damit zur Sicherheit beigetragen haben. Wir bitten Sie darum Probleme die Ihnen auffallen zunächst an info@uib.de zu melden bevor das Problem öffentlich gemacht wird.

12.2 Informiert bleiben

Informationen über Security relevante opsi Updates werden veröffentlicht in:

News Bereich des opsi forums:
https://forum.opsi.org/viewforum.php?f=1

In der opsi Mailingliste:
https://lists.sourceforge.net/lists/listinfo/opsi-announce_de

12.3 Allgemeine Serversicherheit

Die opsi Software auf einem Server kann nicht sicherer als der Server selbst sein. Daher ist es wichtig, dass Sie Ihren *opsi-server* regelmäßig aktualisieren, d.h. die vom Distributor angebotenen Security Updates auch einspielen. Dies gilt sowohl für den *opsi-configserver* wie auch für die *opsi-depotserver*.

Sie können auf dem Server Programme installieren, welche Sie per E-Mail informieren wenn es Paketupdates gibt.

Debian, Ubuntu
: apticron

RHEL, CentOS
: yum-updatesd

Darüber hinaus gibt es viele Möglichkeiten Linuxserver weiter abzusichern. Dies ist aber nicht das Thema dieses Handbuchs. Gerne beraten wir Sie hierzu im Rahmen eines Support- und Pflegevertrages.

12.4 Read Only depotshare

Der von den Clients verwendete *depot_share* sollte read-only sein. Dies ist wichtig um zu unterbinden, dass ein Vireninfizierter Rechner evtl. während der Installation über opsi den Share infiziert und dann der Virus über opsi weiterverbreitet wird.

Seit opsi 4.0.1 gibt es den Share *opsi_depot* welcher read-only ist. Um diesen Share zu verwenden führen Sie bitte folgendes aus:

```
opsi-setup --auto-configure-samba
```

Dieser Befehl erzeugt einen neuen Share *opsi_depot*, welcher read-only ist. Dieser Share verweist auf das Verzeichnis `/var/lib/opsi/depot`. Je nach Distribution des Servers ist dieser Pfad ein symbolischer Link auf `/opt/pcbin/install`.

Damit der neue Share auch von den Clients verwendet wird, müssen Sie auf dem *opsi-configserver* zusätzlich folgendes Script ausführen:

```
for depot in $(opsi-admin -dS method host_getIdents unicode "{\"type\":\"OpsiDepotserver\"}"); do
   echo "Depot: $depot"
   depot_remote=$(opsi-admin -dS method host_getObjects [] "{\"id\":\"$depot\"}" | grep "depotRemoteUrl=" | cut -d "=" \
     -f2)
   depot_local=$(opsi-admin -dS method host_getObjects [] "{\"id\":\"$depot\"}" | grep "depotLocalUrl=" | cut -d "=" -\
     f2)
   depot_remote_new=$(echo $depot_remote | sed "s|/opt_pcbin/install|/opsi_depot|")
   depot_local_new=$(echo $depot_local | sed "s|/opt/pcbin/install|/var/lib/opsi/depot|")
   servertype=$(opsi-admin -dS method host_getObjects [] "{\"id\":\"$depot\"}" | grep "type=" | cut -d "=" -f2)
   opsi-admin -d method host_updateObjects "{\"type\":\"$servertype\",\"id\":\"$depot\",\"depotLocalUrl\":\"\
     $depot_local_new\",\"depotRemoteUrl\":\"$depot_remote_new\"}"
done
```

12.5 Authentifizierung des Clients beim Server

Der Client authentifiziert sich beim Server mit seinem FQDN sowie dem opsi-host-key. Client-seitig liegt dieser Key in der Datei `%programfiles%\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf` und ist nur mit administrativen Rechten lesbar. Serverseitig liegen die opsi-host-keys im jeweiligen Backend (z.B. unter `/etc/opsi/pckeys`).

Zusätzlich zu dieser Authentifizierung kann der opsiconfd angewiesen werden, zu überprüfen ob die IP-Nummer unter der sich der Client meldet auch zu dem angegebenen Clientnamen passt. Um dieses Verhalten zu aktivieren setzen sie in der `/etc/opsi/opsiconfd.conf` folgenden Parameter:

```
verify ip = yes
```

und reloaden den opsiconfd

```
/etc/init.d/opsiconfd reload
```

Achtung
Verwenden Sie diese Feature nur wenn Sie eine vollständig funktionierende Namensauflösung (vorwärts wie rückwärts) für Ihre Clients haben.

12.6 Authentifizierung des Servers beim Client

Seit opsi 4.0.1 gibt es verschieden Möglichkeiten den Client prüfen zulassen ob der Server der kontaktiert vertrauenswürdig ist.

12.6.1 Variante 1: verify_server_cert

Bei der ersten Kontaktaufnahme zu einem opsi-server wird dessen SSL-Zertifikat akzeptiert und unter `C:\opsi.org\opsiclientd\server-certs` abgespeichert. Bei der nächsten Kontaktaufnahme wird der *public key* des gespeicherten Zertifikats ermittelt und mit diesem ein Zufallsstring sowie die eigenen Zugangsdaten verschlüsselt. Diese Daten werden an den Server übergeben. Der Server entschlüsselt mittels des *private key* seines SSL-Zertifikats diese Daten und sendet zum Beweis der erfolgreichen Entschlüsselung den Zufallsstring an den Client zurück. Nach erfolgreicher Prüfung des zurückgegbenen Zufallsstrings erachtet der Client den Server als vertrauenswürdig.

Auf diese Weise kann verhindert werden, dass per Angriff auf das DNS opsi-clients falsche Server als *opsi-server* akzeptieren. Diese Methode minimiert das Risiko, dass bei einem Serverumzug Clients den neuen Server nicht mehr

akzeptieren, da sich die Serverzertifikate von einem auf einen anderen Server übertragen lassen. Außerdem ist keine Verteilung einer certification authority (CA) notwendig.

Die Schwäche dieser Methode ist die Möglichkeit eines *Man-in-the-middle* Angriffs, da aus den oben genannten Gründen das Zertifikat keiner weiteren Prüfung unterzogen wird.

Die Prüfung betrifft die Webservice Kommunikation mit dem *opsi-configserver*.

Wird im Rahmen der opsi WAN-Erweiterung für die WAN-Clients als `clientconfig.depot.protocol webdav` verwendet, so wird auch der *opsi-depotserver* entsprechend geprüft. Abschnitt 15.3.1

Um diese Methode zu aktivieren, muss in der `opsiclientd.conf` in der Sektion *[global]* folgende Option gesetzt werden:

```
verify_server_cert = true
```

Mit dem folgenden Befehl richten Sie den Konfigurationseintrag ein, so das er vom *opsi-configed* bearbeitet werden kann:

```
opsi-admin -d method config_createBool opsiclientd.global.verify_server_cert "verify_server_cert" false
```

Aktivieren können Sie dies nun in dem Sie im *opsi-configed* in der *Serverkonfiguration* oder in den *Hostparameter* einzelner Clients den Wert von *false* auf *true* setzen.

Achtung
Dies sollte nur gemacht werden, wenn man genau weiß, was man tut. Ansonsten ist die Gefahr groß, dass die Clients den Kontakt zum Server verweigern und somit abgehängt sind.

12.6.2 Variante 2: verify_server_cert_by_ca

Dieser Mechanismus funktioniert analog zu der Art und Weise wie Browser die SSL-Zertifikate von Webservern prüfen. SSL-Zertifikate werden klaglos akzeptieren wenn diese auf genau den FQDN (bzw. Wildcard) ausgestellt sind unter dem der Server angesprochen wird und das Zertifikat von einer dem Browser bekannten CA signiert ist.

Der opsicliend enthält eine nur für diesen Zweck erstellte Root-CA der *uib gmbh* und akzeptiert Zertifikate, die durch die CA der *uib gmbh* signiert sind. Ebenso wird überprüft ob der *commonName* mit dem FQDN des Servers übereinstimmt. Ist eine der beiden Bedingungen nicht erfüllt, verweigert der Client die Kommunikation.

Diese Methode ist sicherer als die oben beschriebene, erfordert aber den Erwerb von Zertifikaten über die *uib gmbh*. Die Bedingungen und Preise zum Erwerb solcher Zertifikate erfahren Sie auf der Preisliste der *uib gmbh*. http://uib.de/-www/service_support/support/index.html Die Überschüsse aus dem Zertifikatsverkauf fließen in die Pflege der opsi-Security.

Um diese Methode zu aktivieren, muss in der `opsiclientd.conf` in der Sektion *[global]* folgender Parameter gesetzt werden:

```
verify_server_cert_by_ca = true
```

Mit dem folgenden Befehl richten Sie den Konfigurationseintrag ein, so das er vom *opsi-configed* bearbeitet werden kann:

```
opsi-admin -d method config_createBool opsiclientd.global.verify_server_cert_by_ca "verify_server_cert_by_ca" false
```

Aktivieren können Sie dies nun in dem Sie im *opsi-configed* in der *Serverkonfiguration* oder in den *Hostparameter* einzelner Clients den Wert von *false* auf *true* setzen.

Achtung
Dies sollte nur gemacht werden, wenn man genau weiß, was man tut. Ansonsten ist die Gefahr groß, dass die Clients den Kontakt zum Server verweigern und somit abgehängt sind.

12.7 Authentifizierung beim controlserver des Client

Der *opsiclientd* besitzt eine Webservice-Schnittstelle die es erlaubt dem opsiclientd Anweisungen von aussen zu erteilen (Abschnitt 7.3.9).

Für den Zugriff auf den Webservice wird eine Authentifizierung benötigt. Dies geschieht entweder mittels des lokalen Administrator-Accounts (ein leeres Passwort ist unzulässig) oder mittels leerem Benutzernamen und dem *opsi-host-Schlüssels* als Passwort.

12.8 Konfiguration eines Admin-Networks

Die Idee eines Admin-Networks ist es, administrative Zugriffe auf Server nicht aus dem allgemeinen Produktiv-Netz zu erlauben, sondern nur von einem speziellen und abgesicherten Netzbereich.

Zwar müssen alle *opsi-clients* Zugang zum opsi-webservice haben, diese dürfen aber nur eingeschränkt Daten abrufen und ändern. Ein administrativer Zugang zum Webservice setzt die Mitgliedschaft in der Gruppe *opsiadmin* voraus.

Über die Option `[global] admin networks` in der `/etc/opsi/opsiconfd.conf` kann der administrative Zugriff auf den *opsiconfd* auf Verbindungen von bestimmten Netzwerkadressen eingeschränkt werden.
Es können mehrere Netzwerkadressen durch Kommas getrennt angegeben werden.
Nicht-administrative Client-Verbindungen können auch aus anderen Netzwerken erfolgen.

Der default ist:

```
admin networks = 0.0.0.0/0
```

und erlaubt Zugriff von allen Netzen.

Eine Konfiguration wie z.B.

```
admin networks = 127.0.0.1/32, 10.1.1.0/24
```

würde administrative Zugriffe nur vom Server selbst und aus dem Netz *10.1.1.0/24* erlauben.

12.9 Der user pcpatch

Der user *pcpatch* dient in opsi 4 ausschließlich dem Mounten des depot-Shares durch den Client (und zur Zeit noch dem Schreiben und Lesen von Image-Dateien durch die Netboot Produkte *ntfs-write-image* bzw. *ntfs-restore-image*).

Das Passwort des Benutzers *pcpatch* wird in der Regel verschlüsselt abgelegt und auch nur verschlüsselt übertragen. Es existieren jedoch auch unter gewissen Umständen Möglichkeiten das Passwort in Erfahrung zu bringen. Um den Schaden der hierdurch entstehen kann zu minimieren empfehlen wir folgende Maßnahmen:

In der `/etc/samba/smb.conf` in allen Share-Definitionen ausser *opsi_depot* dem user pcpatch den Zugriff verbieten über den Eintrag:

```
invalid users = root pcpatch
```

Alternativ
In der `/etc/samba/smb.conf` dem User *pcpatch* auf Leserechte beschränken durch den Eintrag in der [global] Sektion:

```
read list = pcpatch
```

Als weitere Maßnahme sollten Sie das Passwort des Users *pcpatch* öfters ändern. Da das Klartext-Passwort niemandem bekannt sein muss, kann es z.B. durch den regelmäßigigen Aufruf (z.B. per cronjob) des folgenden Scriptes auf ein zufälliges Passwort setzen.

pass=$(< /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c16)

```
opsi-admin -d task setPcpatchPassword $(< /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c16)
```

Wenn Sie die Netboot Produkte *ntfs-write-image* bzw. *ntfs-restore-image* nicht verwenden, so können Sie zusätzlich die Anmeldung des users *pcpatch* am Server unterbinden indem Sie in der `/etc/passwd` dem user *pcpatch* die Shell `/bin/false` zuweisen.

13 opsi-backup

13.1 Einführung

Wie jedes andere System auch, sollte das opsi-System auch einem Backup unterzogen werden. Da opsi ein zentrales Werkzeug für das Windows-Client- wie auch das Windows-Server-Management darstellt, sollte der opsi-server gesichert werden. Dieses Handbuch soll einen Einblick in die Backup-Strategie von opsi geben und auch auf Themen, wie das zurückschreiben und das "DisasterRecovery" von opsi.

13.2 Vorbedingungen für ein Backup

Um ein Backup des opsi-Systems anzulegen, gibt es nicht wirklich eine Vorbedingung. Wenn man die zentralen Dateien und Backends des opsi-Systems lokalisiert hat, kann man diese auf diversen Methoden sichern. Die folgende Anleitung soll nicht nur die Frage: "Was soll gesichert werden?" beantworten, sondern auch einen Weg dokumentieren, wie eine Backupstrategie für das opsi-System aussehen könnte.

Das Backupskript sollte als root ausgeführt werden, entweder manuell oder einen root-cronjob, damit man die Konfiguration von opsi lesen kann und auch die Systemkonfiguration feststellen kann. Weiterhin sollte für ein Backup des mysql-Backends das *mysqldump*-Programm installiert sein, dieses findet sich in der Regel in den client-Paketen von mysql.

13.3 Quick Start

Backup erzeugen:

```
opsi-backup create opsi_backup.tar.bz2
```

Erzeugt ein Backup der aktuell genutzten Backends sowie der Konfigurationsdateien im aktuellen Verzeichnis mit dem Namen `opsi_backup.tar.bz2`.

Backup zurück spielen:

```
opsi-backup restore --backends=all --configuration opsi_backup.tar.bz2
```

Stellt die Daten aus dem Backupfile `opsi_backup.tar.bz2` aus dem aktuellen Verzeichnis wieder her.

13.4 Elementare Teile von opsi

Opsi kann man grob in fünf Teile gliedern. Die folgenden fünf Teile sind opsi spezifisch und können von System zu System, je nach Konfiguration variieren.

13.4.1 Opsi Konfiguration

Der mit Abstand wichtigste Teil von opsi, ist die Konfiguration. Getreu nach LSB (Linux Standard Base) befindet sich die Konfiguration von opsi unter `/etc/opsi`.

Dieses Verzeichnis beinhaltet hauptsächlich die Backend-Konfiguration, die Webservice-Konfiguration und das SSL-Zertifikat für den Webservice. Weiterhin ist hier auch das Backend-Extend untergebracht, die Konfiguration des opsipxeconfd, des opsi-product-updater und auch die modules-Datei, die Ihnen Ihre kofinanzierten Module freischaltet.

Um später ein volles DisasterRecovery zu verwirklichen, muss das Verzeichnis `/etc/opsi` gesichert werden.

Die Sicherung hat neben dem DisasterRecovery noch einen weiteren Vorteil: Wenn man viele Konfigurationen von opsi geändert hat und das System nicht mehr richtig arbeitet, ist ein Rücksprung auf eine vorherige funktionierende Version meist leichter und schneller, als die Fehlersuche.

Dieser Bereich wird mit `opsi-backup` gesichert.

13.4.2 Opsi Backends

Im folgenden Kapitel werden die Backends von opsi aufgezählt. Diese bilden das Herzstück der opsi-Datenhaltung. Alle Clients, Produkte, Konfigurationen, Statis, etc... sind in der jeweiligen Datenhaltung abgelegt.

Opsi bietet folgende Datenbackends:

Tabelle 1: opsi-Backends

Backend	**Beschreibung**
file-Backend	Backend auf Dateibasis, momentan der default bei opsi
mysql-Backend	Volle mysql-Backend Unterstützung seit opsi 4.0
ldap	komplette Datenhaltung in einem LDAP-Verzeichnis
univention	angepasstes LDAP Backend für UCS-Systeme
dhcp	spezial Backend bei Verwendung von des dhcpd auf dem opsi-server

Wenn Sie nicht wissen, welches Backend sie einsetzen, setzen Sie wahrscheinlich das file-Backend ein. opsi ist aber auch dafür ausgelegt, mehrere Backends gleichzeitig an zu setzen. Welche Backends, für welche Funktionen von opsi eingesetzt werden, wird in der `/etc/opsi/backendManager/dispatch.conf` konfiguriert.

Dieser Bereich wird mit `opsi-backup` gesichert.

13.4.3 Opsi Depotfiles

Die Depotfiles sind deshalb interessant, da Sie die eigentlichen Dateien der zu verteilenden Software enthalten. Die Localboot-Produkte, wie auch die Netboot-Produkte haben Ihre Files jeweils unterhalb von `/opt/pcbin/install`. bzw. `/var/lib/opsi/depot`.

Je nachdem, wie viel Software auf dem opsi-server vorgehalten wird und wie viele Betriebssystem-Installationen inklusive Treibern vorgehalten werden, kann dieses Datenvolumen enorme Ausmaße annehmen.

Es gibt verschiedene Ansätze diese Dateien zu sichern. Die einfachste Alternative ist das *Rsnapshot*. Es gibt aber elegantere Lösungen, wie das Verlegen dieser Daten in redundant ausgelegte Filesysteme auf einem SAN, etc.

Dieser Bereich wird mit `opsi-backup` **nicht** gesichert.

13.4.4 Opsi Workbench

Der Bereich opsi Workbench, welcher auch als gleichnamige Samba-Freigabe (*opsi_workbench*) in opsi eingesetzt wird, beinhaltet die Stände der eigenen Software-Paketierung. Das Verzeichnis ist standardmäßig unter `/home/opsiproducts` Wenn dieser Share, wie vorgesehen dafür verwendet wird, um eigene Pakete in verschiedenen Revisionen dort vor zu halten, sollte dieses Verzeichnis auch gesichert werden.

Auch hier bietet sich das Tool *rsnapshot* an.

Dieser Bereich wird mit `opsi-backup` **nicht** gesichert.

13.4.5 Opsi Repository

Das Verzeichnis unter `/var/lib/opsi/repository` wird dazu verwendet, um opsi-Pakete zu puffern. Anders als die opsi Workbench, dient es aber nicht dem Paketieren von opsi Paketen, sondern die opsi Pakete welche dort abgelegt werden, sollen vorgehalten werden, um eventuell das Synchronisieren auf anderen Servern, oder das Synchronisieren mit dem `opsi-product-updater` zu vereinfachen.

Diese Dateien sind für ein DisasterRecovery nicht unbedingt nötig, können aber auch mit dem Tool *rsnapshot* gesichert werden.

Dieser Bereich wird mit `opsi-backup` **nicht** gesichert.

13.5 Das `opsi-backup` Programm

Mit dem Kommandozeilenprogramm `opsi-backup` existiert ein Werkzeug, dass die Erstellung und das Wiederherstellen einfacher Backups kompfortabel erledigt.

Dazu lässt sich `opsi-backup` mit drei grundlegenden Befehlen steuern: `create`, `restore` und `verify`.
Die Option `--help` gibt einen detaillierten Überblick über alle Optionen, die `opsi-backup` akzeptiert.
Ein mit `opsi-backup` erstelltes Backup ist ein Rohbackup, dass bedeutet, es werden keine Datein auf logischer Ebene gesichert, sondern es werden Sicherungen der in den Backends abgelegten Datein in den entsprechenden Strukturen angefertigt.
Ein solches Backup lässt sich daher auch nur für eine identische Backendkonfiguration zurückspielen.

Ein mit `opsi-backup` erstelltes Backup ist immer ein Vollbackup (`opsi-backup` unterstützt keine incrementellen oder differenziellen Backups).

Zu beachten ist, dass `opsi-backup` keine Sicherung der Depot Dateien, der Workbench Dateien sowie der Repository Dateien durchführt. Diese Datein sollten daher anderweitig gesichert werden.

Der mit `opsi-backup` erstellte backup file ist eine komprimierte tar Datei, deren Inhalt sich entsprechend auch anschauen lässt.

```
opsi-backup --help
```

Achtung
Ein Backup, dass mit `opsi-backup` erstellt wird, kann unter anderem Passwörter und PC-Keys enthalten, und sollte daher entsprechend sicher archiviert werden.

13.5.1 Ein Backup anlegen

Das Anlegen eines neuen opsi Backups erfolgt mit dem Befehl `opsi-backup create`. Wird dieser Befehl ohne weitere Parameter angegeben erstellt das Programm ein Archiv mit allen Daten der Backends sowie der Konfiguration. Der Dateiname wird dabei automatisch generiert. Für den Befehl `opsi-backup create` sind zusätzliche Programmhilfen verfügbar, welche über die Option `--help` ausgegeben werden.

```
opsi-backup create
opsi-backup create --help
```

Es ist auch möglich, den Dateinamen oder das Zielverzeichnis des neuen Backups vorzugeben. Dazu wird einfach ein Dateiname oder ein Zielverzeichnis einfach an den entsprechenden Befehl angehängt. Wird ein Verzeichnis übergeben, generiert `opsi-backup` automatisch einen Dateinamen in diesem Verzeichnis. Ein durch `opsi-backup` generierter Dateiname hat die Form `<hostname>_<opsi-version>_<datum>_<uhrzeit>` und ist daher gut zur Archivierung meherere Backups geeignet. Wird ein Dateiname fest forgegeben, so wird ein älteres Backup mit dem selben Namen durch `opsi-backup` überschrieben.

```
opsi-backup create /mnt/backup/opsi_backup.tar.bz2
opsi-backup create /mnt/backup/
```

Zusätzlich ermöglicht das `create` Kommando die Steuerung des Backups mittels der folgenden Optionen:

- `--backends {file,mysql,dhcp,all,auto}`
  Ermöglicht die Auswahl der Backends, die in dem Backup eingeschlossen werden sollen. Diese Option kann mehrfach angegeben werden, um mehrere Backends anzugeben. Die Option `--backends=all` steht für alle Backends. Die Voreinstellung (default) für diese Optionen ist `--backends=auto`, was dafür sorgt, dass `opsi-backup` versucht, die verwendeten Backends anhand der Konfigurationsdatei `/etc/opsi/backendManager/dispatch.conf` zu ermitteln. Im Moment werden folgende Backends unterstützt: `mysql`, `file`, `dhcp`

```
opsi-backup create --backends=file --backends=mysql
opsi-backup create --backends=all
```

Tipp
Wenn Sie ein nicht unterstütztes Backend (wie z.B. *ldap*) verwenden, so können Sie vor dem Backup dieses mit dem Befehl `opsi-convert` in ein Backend konvertieren, dass sich per `opsi-backup` sichern lässt.

- `--no-configuration`
  Schließt die Opsi Konfiguration aus dem Backup aus.

```
opsi-backup create --no-configuration
```

- `-c [{gz,bz2,none}], --compression [{gz,bz2,none}]`
  Spezifiziert die Kompressionsmethode, mit der das Archiv komprimiert werden soll. `none` steht hier für nicht komprimieren, die Standardkompression (default) ist `bz2`.

```
opsi-backup create -c bz2
```

- `--flush-log`
  Die Sicherung des mysql-Backends erfolgt intern über einen *mysqldump* Befehl. Das bedeutet, dass die Daten genauso gesichert werden, wie die Datenbank sie zu diesem Zeitpunkt sieht (unabhängig davon ob die Daten schon auf Platte stehen oder nur im Speicher). Somit ist das erstellte Backup evtl. aktueller und unterscheidet sich vom Stand der Datenbankdateien. Möchte man dies vermeiden, so müssen die von mysql im Speicher gehaltenen Daten vorher auf die Festplatte geschrieben werden. Ist die Option `--flush-log` angegeben, wird `opsi-backup` versuchen, diese Operation durchzuführen (also die Daten aus dem Speicher auf die Platten zuschreiben). Allerdings benötigt der entsprechende Datenbankuser der opsi Datenbank dazu die entsprechende MySQL Berechtigung RELOAD. Standardmäßig wird der opsi Benutzer aber ohne dieses Recht angelegt! Besitzt er diese nicht (und die Option `--flush-log` ist angegeben) wird das Backup fehlschlagen. Verwenden Sie daher diese Option nur, wenn Sie vorher die Rechte des Datenbankusers angepasst haben.

```
opsi-backup create --backends=mysql --flush-log
```

Beispiel

```
opsi-backup create --no-configuration --backends=all opsi_backup.tar.bz2
```

13.5.2 Backups archivieren

Von Haus aus bringt `opsi-backup` keine Funktionen zum Archivieren von Backups mit. Der Administrator hat daher Sorge zu tragen, dass erzeugte Backups sicher und versioniert ablegt werden. Außerdem löscht `opsi-backup` niemals selbstständig ältere Backup Version (außer sie werden mittels `create` überschrieben). Da `opsi-backup` immer Vollbackups und keine inkrementellen Backups anlegt, kann es schnell zu großen Datenmengen kommen. Hier muss ebenfalls der Administrator sorge tragen, dass ältere Backups wenn nötig regelmässig gelöscht werden.

13.5.3 Ein Backup verifizieren

Mit dem Befehl `opsi-backup verify` kann das Archiv auf interne Integrität geprüft werden. Diese Prüfung ist keine logische Prüfung der Daten, es handelt sich um eine reine Prüfung auf die Korrektheit der im Archiv gespeicherten Daten. Für den Befehl `opsi-backup verify` sind zusätzliche Programmhilfen verfügbar, welche über die Option `--help` ausgegeben werden.

Beispiel

```
opsi-backup verify opsi_backup.tar.bz2
opsi-backup verify --help
```

Tipp
Wird der Befehl `opsi-backup verify` explizit auf der Konsole aufgerufen ist es häufig sinnvoll, die `opsi-backup` Standardausgabe zu aktivieren: `opsi-backup -v verify opsi_backup.tar.bz2`

13.5.4 Ein Backup wiederherstellen

Das Wiederherstellen des Archivs erfolgt mit dem Befehl `opsi-backup restore`. Dabei werden (per default) die Backends anhand der aktuellen Konfiguration eingespielt. Es kann also kein reines Backend Backup wiederhergestellt werden, ohne dass eine opsi Konfiguration vorhanden ist. Der Befehl `opsi-backup restore` braucht als Parameter das Backup Archiv, aus dem Daten wiederhergestellt werden. Für den Befehl `opsi-backup restore` sind zusätzliche Programmhilfen verfügbar, welche über die Option `--help` ausgegeben werden.

`opsi-backup restore` akzeptiert folgende Optionen:

- `--backends {file,mysql,dhcp,auto,all}`
  Stellt das spezifizierte Backend wieder her. Diese Option kann mehrfach angegeben werden, um mehrere Backends anzugeben. Die Option `--backends=all` steht für alle Backends. Als Voreinstellung (default) wird die Option `--backends=auto` verwendet, was dazu führt, dass `opsi-backup` versucht, anhand der Konfigurationsdatei */etc/opsi/backendManager/dispatch.conf* festzustellen, welche Backends wiederherzustellen sind.

```
opsi-backup restore --backends=file --backends=mysql opsi_backup.tar.bz2
opsi-backup restore --backends=all opsi_backup.tar.bz2
```

Achtung
Wenn Sie seit der Erstellung des Backups das Backend gewechselt haben, so wird die dafault Einstellung keine Daten zurück sichern.

- `--configuration`
  Stellt die Opsi Konfiguration wieder her. Diese Option ist beim `restore` Vorgang kein default.

```
opsi-backup restore --configuration opsi_backup.tar.bz2
```

- `-f, --force`
  `opsi-backup` führt vor dem Wiederherstellen eines Backups, eine Sicherheitsprüfung durch, um zu überprüfen, ob die aktuelle opsi Installation mit der Installation des Backups übereinstimmt (opsi Version, OS-Version, Host- und Domain Name). Mit dieser Option lässt sich diese Prüfung umgehen.

```
opsi-backup restore -f opsi_backup.tar.bz2
```

Beispiel

```
opsi-backup restore --configuration --backends=all opsi_backup.tar.bz2
```

14 opsi-Lizenzmanagement

14.1 Vorbedingungen für die opsi Lizenzmanagement Erweiterung

Dieses Modul ist momentan eine kofinanzierte opsi Erweiterung. Das bedeutet die Verwendung ist nicht kostenlos. Weitere Details hierzu finden Sie in Abschnitt 6.

14.2 Überblick

14.2.1 Funktion und Features

Das opsi-Lizenzmanagement-Modul ist darauf ausgerichtet, die aufwändige und komplexe Verwaltung von Lizenzen für die diversen nicht-freien Softwareprodukte, die auf mit opsi verwalteten Clients eingesetzt werden, zu vereinheitlichen und zu vereinfachen.

Die wesentlichen Features sind:

- Handhabung der Lizenzverwaltung innerhalb der gleichen Oberfläche wie Softwareverteilung und Betriebssysteminstallation, d.h. im opsi-Konfigurationseditor.
- Automatische Bereitstellung, Zuteilung und Reservierung der Lizenzkeys.
- Verfügbarkeit verschiedener Lizenzmodelle
  - Standard-Einzellizenz,
  - Volume-Lizenz (1 Lizenzkey - eine bestimmte Zahl von Installationen) oder Campus-Lizenz (1 Schlüssel - unbegrenzte Zahl von Installationen)
  - PC-gebundene Lizenz,
  - Concurrent License (Lizenzserver-vermittelt)
- Freigabe der Lizenzkeys bei der Deinstallation von Software.
- Manuelle Bearbeitung der Lizenzzuordnungen z.B. für Lizenzen von Software, die nicht mit opsi verteilt werden.
- Report-Funktion auch für nicht mit opsi verwaltete Lizenznutzungen auf der Basis der Software-Inventarisierung und Abgleich mit den opsi-basierten Nutzungen.

14.2.2 Aufruf der Lizenzmanagement-Funktionen im *opsi-configed*

Der *opsi-configed* verfügt für das Lizenzmanagement über ein gesondertes Fenster.
Es ist über die Schaltfläche "Lizenzen" im Hauptfenster des Konfigurationseditors erreichbar, sofern das Lizenzmanagement-Modul in der aktuellen opsi-Konfiguration aktiv ist (vgl. den Eintrag für "license management" im Hauptmenü unter */Hilfe/Module*).
Bei nicht aktiviertem Lizenzmanagement wird lediglich ein Hinweis angezeigt.

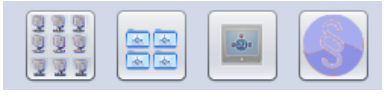


Abbildung 61: opsi-configed: Leiste mit Schaltfläche "Lizenzen" (rechts)

Das Modul opsi-Lizenzmanagement ist als ein kofinanziertes opsi-Erweiterungsprojekt realisiert. Das bedeutet, dass es - bis zur Refinanzierung - per entsprechender *modules*-Konfiguration entweder zu Testzwecken für einen begrenzten Zeitraum oder dauerhaft für die opsi-Anwender aktiviert ist, die den festgesetzten Beitrag zur Entwicklung eingezahlt haben.

14.3 Lizenzpools

14.3.1 Was ist ein Lizenzpool?

Für jede Art von benötigten Lizenzen ist im opsi-Lizenzmanagement ein *Lizenzpool* (*license pool*) einzurichten.

Der Lizenzpool ist dabei ein Konstrukt, das die gedankliche Zusammenfassung aller Lizenzen beschreibt, die für eine bestimmte Art von installierter oder zu installierender Software vorrätig sind.

Dieses Konstrukt steht im Mittelpunkt aller Aktivitäten im opsi-Lizenzmanagement.

Demgemäß ist die erste Tab-Seite des Lizenzmanagement-Fensters im *opsi-configed* der Administration der Lizenzpools gewidmet.

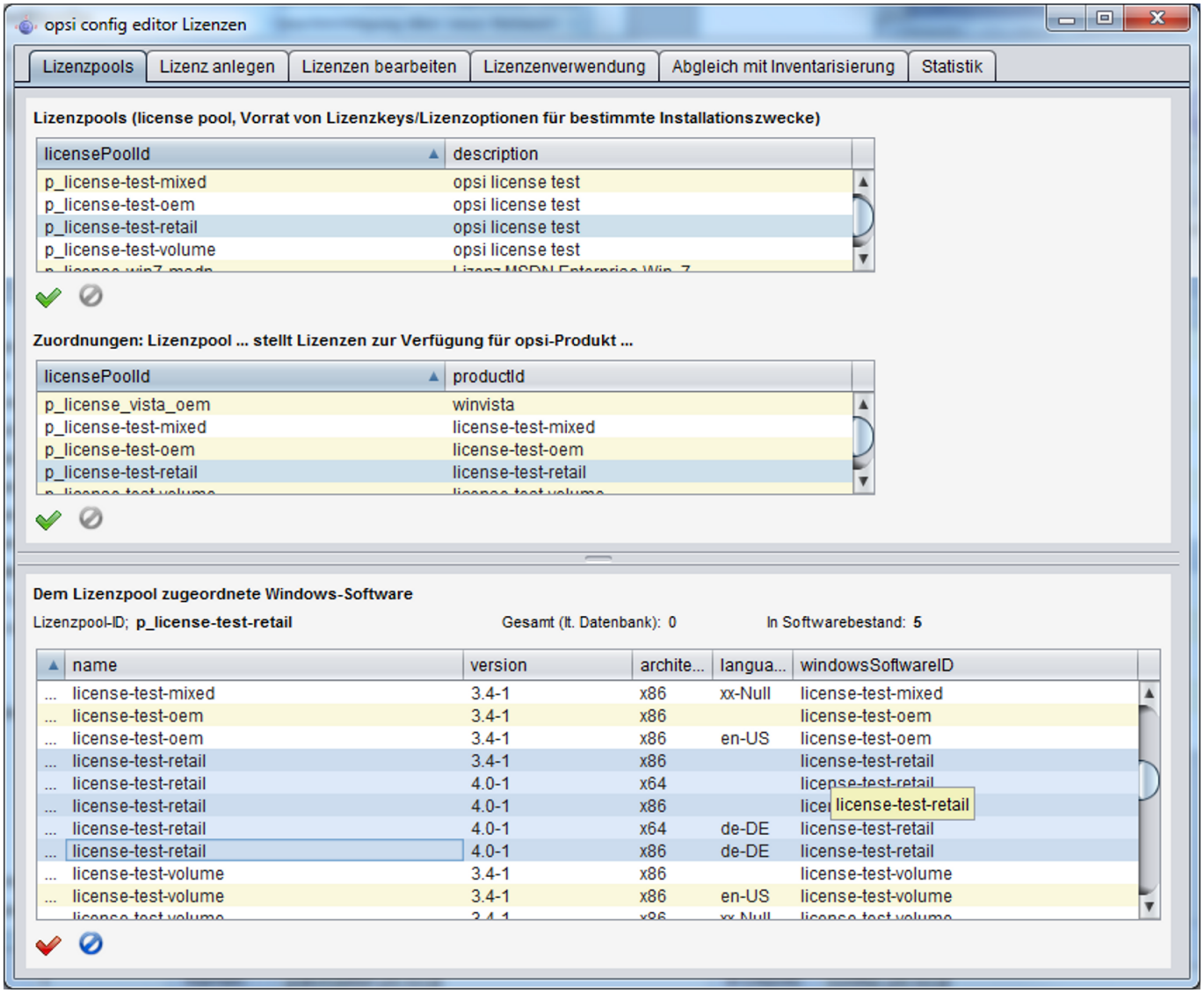


Abbildung 62: Lizenzmanagement:Tab Lizenzpools

14.3.2 Verwaltung von Lizenzpools

Auf der Lizenzpool-Seite befindet sich im oberen Bereich die Tabelle der verfügbaren Lizenzpools.

Hier ist das Feld *description* editierbar.

Weitere Bearbeitungsfunktionen erschließen sich über das Kontextmenü, am wichtigsten: das Erzeugen eines neuen Lizenzpools.

Nach dem Einfügen einer neuen Zeile in die Tabelle muss eine (eindeutige) *licensePoolId* in das entsprechende Feld eingetragen werden, z.B. *softprod_pool*. Bitte dabei keine Umlaute etc. verwenden. Eingegebene Großbuchstaben werden beim Speichern automatisch in Kleinbuchstaben konvertiert.

Die ID kann nur bis zum ersten Speichern noch bearbeitet werden. Als Schlüssel des Datensatzes ist sie danach unveränderlich.

In der Maske aktiviert jeder Bearbeitungsvorgang die Statusanzeige in der Art, dass die Farbe des O.K.-Buttons (Häkchen) von grün nach rot wechselt und auch der Cancel-Button seinen aktiven Modus annimmt. Durch Betätigen des betreffenden Buttons (oder mittels Kontextmenü) kann die Veränderung dann permanent gemacht (gespeichert) bzw. widerrufen werden.

14.3.3 Lizenzpools und opsi-Produkte

Im Standardfall gehört zu einem opsi-Produkt, das ein lizenzpflichtiges Software-Produkt installiert (z.B. den *Acrobat Writer*), genau ein Lizenzpool, aus dem die benötigten Lizenzen geschöpft werden.

Weniger übersichtlich ist die Situation, wenn ein opsi-Produkt mehrere lizenzpflichtige Software-Produkte installiert, etwa wenn zu einem Paket "Designerprogramme" sowohl *Adobe Photoshop* wie auch *Acrobat Writer* gehören sollen.

Das opsi-Produkt muss dann Lizenzen aus mehreren Pools anfordern. Da es gleichzeitig auch weitere opsi-Produkte geben kann, die z.B. Lizenzen aus dem Pool für den *Acrobat Writer* benötigen, kann auch die umgekehrte Beziehung, vom Lizenzpool zum opsi-Produkt, gelegentlich mehrdeutig sein. Derartige Mehrdeutigkeiten können natürlich durch eine entsprechende Policy beim Produktebau vermieden werden.

Im zweiten Abschnitt der Lizenzpool-Seite wird die Tabelle aller Zuordnungen zwischen Lizenzpools und den productIds von opsi-Produkten dargestellt.

Wie in allen anderen Tabellen des Lizenzmanagements wird durch einen Klick auf einen Spaltentitel die Tabelle nach dem Wert in der betreffenden Spalte umsortiert;
nochmaliges Klicken ändert die Sortierungsrichtung.
Die Sortierung kann genutzt werden, um alle Zuordnungen von opsi-Produkten zu einem Lizenzpool zusammenhängend darzustellen oder umgekehrt alle einem opsi-Produkt zugeordneten Lizenzpools zu erkennen.

Über das Kontextmenü ist wieder die Funktion erreichbar, mit der eine neue Tabellenzeile, hier also eine neue Zuordnung Lizenzpool-Produkt-ID, erstellt werden kann. Zur Eingabe von Lizenzpool-ID und Produkt-ID wird bei Klick in das Tabellenfeld jeweils die Liste der verfügbaren Werte angezeigt, aus der ein Wert ausgewählt werden kann.

14.3.4 Lizenzpools und Windows-Software-IDs

Die dritte Tabelle der Seite "Lizenzpools" stellt in Form einer Mehrfachauswahl dar, welche IDs aus der Softwareinventarisierung dem in der ersten Tabelle markierten Lizenzpool zugeordnet sind.

Eine (Windows-Software-) ID ist ein eindeutiger Schlüsselwert, welcher im Rahmen des opsi-Software-Audits ermittelt und an den Server übertragen wird. Die Zuordnungen zu Lizenzpools können bearbeitet werden, indem die Mehrfachauswahl verändert wird (wie üblich durch Strg-Mausklick bzw. Shift-Mausklick).

Das Kontextmenü der Tabelle bietet die Option, zwischen der Anzeige nur der aktuell zugeordneten IDs oder sämtlicher im Software-Audit erfassten IDs umzuschalten.

Die Zuordnung zwischen Windows-Software-IDs und Lizenzpools dient im Lizenzmanagement vor allem dazu, nicht nur die Installationen per opsi-Setup, sondern auch die Gesamtzahl der faktischen Installationen einer Software (wie aus der Registry der Clients ausgelesen) zu kontrollieren und abzugleichen mit der Zahl der dokumentiert verfügbaren Lizenzen eines Lizenzpools (Tab "Statistik", s. unten).

14.4 Einrichten von Lizenzen

Das Einrichten einer Lizenz bzw. die Bereitstellung einer Lizenz in einem Lizenzpool, erfordert mehrere Schritte. Sie können, mit vorgegebenen Optionen vorstrukturiert, auf der zweiten Tab-Seite des Lizenzmanagement-Fensters (Titel "Lizenz anlegen") durchgeführt werden.

Die Seite startet mit einer (hier nicht editierbaren) Tabelle der verfügbaren Lizenzpools. Dort ist zunächst der Pool auszuwählen, für den eine Lizenz eingerichtet werden soll.

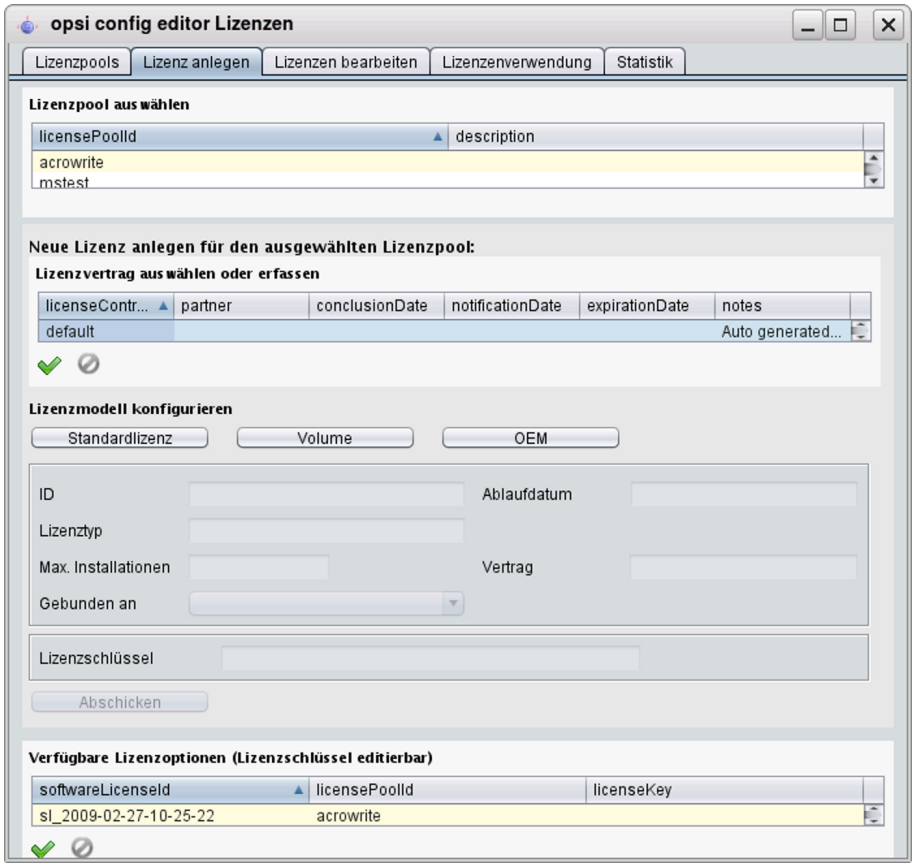


Abbildung 63: Lizenzmanagement:Tab "Lizenz anlegen"

Bevor die weitere Schritte beschrieben werden, empfiehlt es sich, einige Begrifflichkeiten zu klären:

14.4.1 Aspekte des Lizenzkonzepts

Unter **Lizenzierung** *(licensing)* soll die faktische Zuweisung der Erlaubnis zur Nutzung einer Software (durch Installation einer Software) verstanden werden. Sie schließt oft, aber nicht notwendig die Nutzung eines hierfür bestimmten **Lizenzschlüssels** *(license key)* ein.

Das **Lizenzierungsrecht** ist die in ihrem Geltungsumfang definierte Erlaubnis, solche Zuweisungen durchführen zu dürfen. In der opsi-Datenbank wird das Lizenzierungsrecht als *software license* bezeichnet, ein entsprechender Datensatz ist demgemäß identifiziert durch eine *softwareLicenseId*. Verschiedene Varianten der konkreten Ausgestaltung des Lizenzierungsrechts (z.B. für wie viele PCs, mit welcher Gültigkeitsdauer etc.) werden als **Lizenzmodelle** bezeichnet. Ein Lizenzierungsrecht gründet in einem **Lizenzvertrag** *(license contract)*, der es im juristischen Sinn feststellt und dokumentiert.

Eine **Lizenzierungsoption** definiert die Anwendungsmöglichkeit eines Lizenzierungsrechts für einen bestimmten Lizenzpool. In opsi ist die Lizenzierungsoption festgelegt durch die Kombination einer *softwareLicenseId* und einer

licensePoolId. Zur Lizenzierungsoption gehört auch der Wert eines spezifischen Lizenzschlüssels (*licenseKey*, sofern er für eine Installation erforderlich ist).

Schließlich dokumentiert eine **Lizenznutzung** die "gezogene" Lizenzierungsoption, d.h. die erfolgte Anwendung einer Lizenzierungsoption für einen Client. Sie ist die vollzogene und berechtigte Lizenzierung einer Softwareinstallation. Beschrieben wird sie durch die Kombination *softwareLicenseId*, *licensePoolId* und dem eindeutigen Namen des betreffenden Clients, *hostId*. Der verwendete Lizenzschlüssel (*licenseKey*) wird ergänzend notiert.

14.4.2 Lizenzvertrag erfassen

Nach der Auswahl des Lizenzpools, für den eine Lizenzierungsoption angelegt werden soll, ist im zweiten Schritt der Lizenzvertrag zu bestimmen, auf den die Lizenzierung letztlich gründen soll. Im Seitenabschnitt "Lizenzvertrag auswählen oder erfassen" auf der Tab-Seite "Lizenz anlegen" kann ein vorhandener spezifischer Vertrag in der Tabelle ausgewählt oder ein neuer Vertrags-Datensatz angelegt werden.

In einem Lizenzvertrags-Datensatz werden wichtige Ordnungsgesichtspunkte für einen Vertrag in den Feldern (Vertrags-) *partner*, Abschlussdatum (*conclusion date*), Benachrichtigungsdatum (*notification date*) und Auslaufdatum (*expiration date*) dokumentiert. Hinzu kommt ein freies Notizfeld (*notes*), um z.B. den Aufbewahrungsort für das Realdokument eines Vertrages aufzunehmen. Die Vertrags-ID (*licenseContractId*) dient zur Identifizierung des Lizenzvertrags in der Datenbank.

Die Erfassung eines neuen Datensatzes wird über das Kontextmenü gestartet. Es werden automatisch Standard-Einträge generiert, insbesondere eine aus der aktuellen Zeit generierte Vertrags-ID und als Vertragsabschlussdatum der aktuelle Tag. Wenn die Vertragsbedingungen sich z.B. aus einem Software-Kauf implizit ergeben bzw. anderweitig dokumentiert und verfolgt werden können, können die Standard-Einträge belassen werden. Andernfalls sind hier Werte einzugeben, die eine geordnete Verfolgung des zugrundeliegenden Vertrags z.B. durch Verweis auf ein Aktenzeichen im Feld *notes* erlauben.

Die Vertrags-ID kann nur bearbeitet werden, solange der Datensatz nicht gespeichert ist.

14.4.3 Lizenzmodell konfigurieren

Der dritte Seitenabschnitt der Tab-Seite "Lizenz anlegen" dient dazu, die Ausgestaltung des einzurichtenden Lizenzierungsrechts festzulegen.

Es werden verschiedene Varianten angeboten:

- Standardlizenz
- Volumen-Lizenz
- OEM-Lizenz
- Concurrent-Lizenz

Jede Option ist durch einen Button repräsentiert, bei dessen Betätigung die Felder im folgenden Formularbereich vor ausgefüllt werden.

Standardlizenz soll bedeuten, dass die Lizenz zu einer Einzel-Installation der Software berechtigt und diese auf einem beliebigen PC erfolgen kann. Ein ggf. erfasster Lizenzschlüssel wird nur für eine Installation verwendet.

Eine **Volumen-Lizenz** legitimiert n Installationen, ggf. mit ein- und demselben Lizenzschlüssel. $n = 0$ soll dabei bedeuten, dass innerhalb des Netzes der Schlüssel beliebig oft zu Installationen verwendet werden darf (**Campus-Lizenz**).

Als **OEM-Lizenz** wird die Situation bezeichnet, dass eine Lizenz nur für einen, festzulegenden PC genutzt werden darf. Dies ist häufig die intendierte Lizenzart, wenn ein PC mit vorinstalliertem Betriebssystem gekauft wird.

Die **Concurrent-Lizenz** ist aus opsi-interner Sicht eine Volumenlizenz, die beliebig häufig genutzt werden darf. Mit der Auszeichnung des Lizenzmodells als Concurrent-Lizenz ist nach außen jedoch die Aussage verbunden, dass die

Anzahl der faktisch in Anspruch genommenen Lizenzierungen auf andere Weise kontrolliert wird, z.B. durch einen Lizenzserver.

Wenn einer der Buttons betätigt wird, erhält auch das ID-Feld eine Vorschlagsbelegung mit einem auf der Basis von Datum und Zeit generierten String, der bearbeitet werden kann.

Je nach Lizenztyp können die anderen Felder editiert werden oder sind unveränderlich.

Das Feld "Ablaufdatum" definiert die technische Gültigkeitsgrenze des Lizenzierungsrechts (während das inhaltlich gleichbedeutende Feld *expirationDate* der Lizenzvertragstabelle Dokumentationszwecken dient). Es ist allerdings nur für einen künftigen Gebrauch vorgesehen, eine Verwendung ist derzeit nicht implementiert.

14.4.4 Abschicken der Daten

Der Button "Abschicken" veranlasst, dass die erfassten Daten an den opsi-Service gesendet und - sofern kein Fehler auftritt - permanent in die opsi-Datenhaltung überführt werden.

Dabei werden Datensätze für ein Lizenzierungsrecht (*software license*) basierend auf dem ausgewählten Vertrag und eine darauf bezogene Lizenzierungsoption erzeugt.

Die Liste der verfügbaren Lizenz(ierungs)optionen, die im unteren Seitenabschnitt dargestellt ist, wird automatisch neu geladen und die Markierung auf die neu erzeugte Option gesetzt.

An dieser Stelle kann, falls erforderlich, der erfasste Lizenzschlüssel korrigiert werden.

14.5 Lizenzierungen bearbeiten

In neunzig Prozent der Anwendungsfälle werden die Eingabe- und Editiermöglichkeiten der Tab-Seiten "Lizenzpools" und "Lizenz anlegen" genügen, um Lizenzoptionen zu erfassen und zu editieren.

Weitere Details der Lizenzkonfiguration macht die Tab-Seite "Lizenzen bearbeiten" zugänglich. Sie präsentiert die Interna der Lizenzierungsoptionen in drei Tabellen und erlaubt ggf. deren Anpassung an spezifische Erfordernisse.

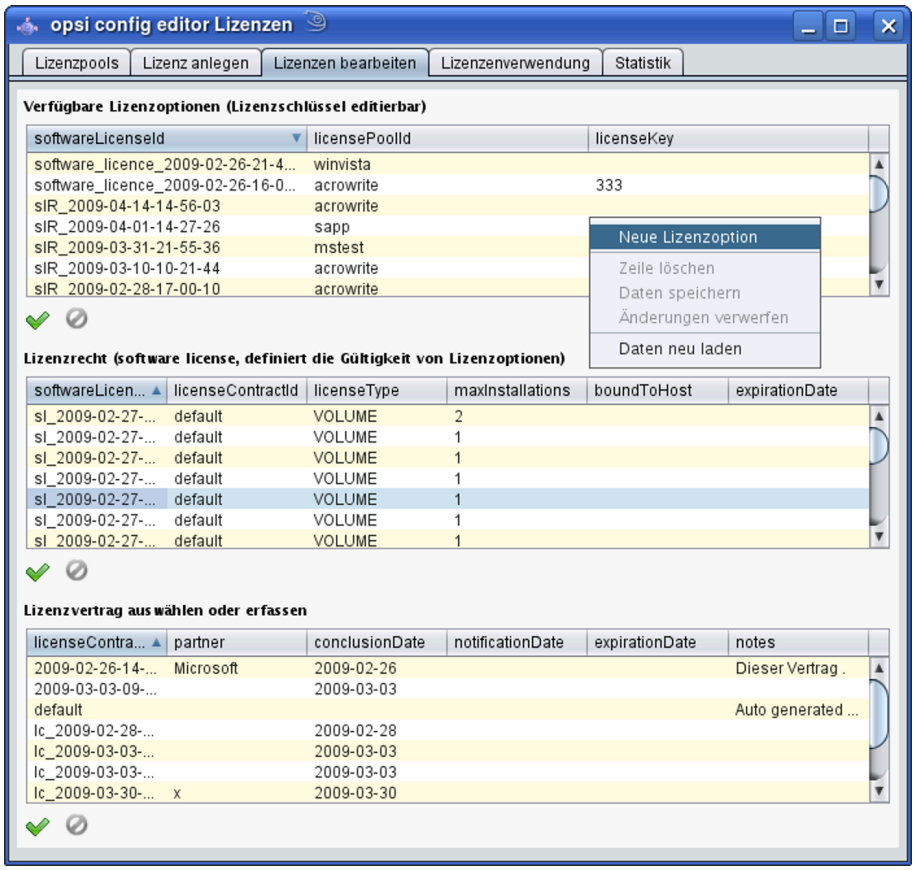


Abbildung 64: Lizenzmanagement:Tab "Lizenzierungen bearbeiten"

Im folgenden Abschnitt wird gezeigt, wie eine Lizenz mit Downgrade-Option konfiguriert werden kann, wie sie z.B. von Microsoft beim Kauf einer Windows-7-Professionallizenz angeboten wird.

14.5.1 Beispiel Downgrade-Option

Die Downgrade-Option bedeutet, dass anstelle der gekauften Software auch die entsprechende Vorgängerversion, z.B. Windows XP anstelle von Windows Vista, installiert werden darf. Bei diesem Microsoft-Modell darf irgendein für die Vorgängerversion vorhandener Lizenzschlüssel für eine zusätzliche Installation verwendet werden, für die er ursprünglich nicht legitimiert war.

Im opsi-Modell kann diese Konstruktion folgendermaßen abgebildet werden:

Auf der Tab-Seite "Lizenz anlegen" wird die Vista-Lizenz regulär erfasst. Das Ergebnis der Prozedur ist eine neue Lizenzierungsoption (angezeigt in der entsprechenden Tabelle am Seitenende), die auf einem gleichfalls neu angelegten Lizenzierungsrecht beruht. Letzterer ist identifizierbar durch den Wert von *softwareLicenseId*.

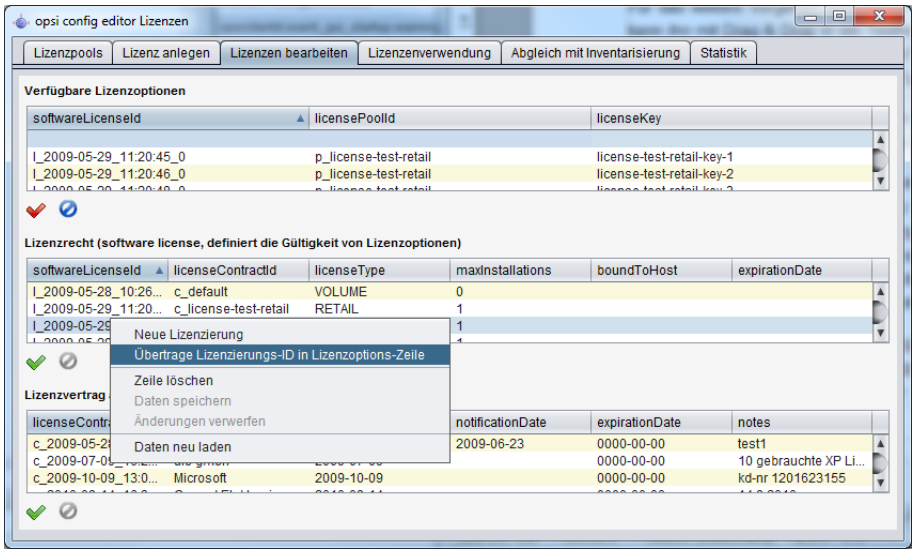


Abbildung 65: Lizenzmanagement:Lizenzmanagement:Kopieren der License-ID in die Lizenzoptionen über das Kontext-Menü

Für das weitere Vorgehen wird dieser Wert benötigt. Man kann ihn sich merken oder kann einen Editor als Zwischenablage nutzen und ihn dorthin mit *Drag & Drop* übertragen. Oder man sucht ihn auf der Tab-Seite "Lizenzen bearbeiten" in der dortigen Tabelle der Lizenzierungsrechte wieder heraus (bitte das Kontextmenü der Tabelle beachten: hier findet sich eine Spezialfunktion zum Kopieren der ID).

Der entscheidende Schritt besteht nun darin, eine Verknüpfung des gegebenen Lizenzierungsrechts mit einem zusätzlichen Lizenzpool herzustellen.

Dazu ist auf der Tab-Seite "Lizenzen bearbeiten" in der Tabelle der verfügbaren Lizenzoptionen ein neuer Datensatz anzulegen. In die betreffenden Felder des Datensatzes sind die ID des Lizenzierungsrechts, die *softwareLicenseId*, sowie die ID des zusätzlichen Lizenzpools - im Beispiel die für Windows XP - einzutragen. Für die Installation von Windows XP ist zusätzlich ein hierfür geeigneter Schlüssel, z.B. ein bei einem anderen Client bereits verwendeter, hinzuzufügen.

Nach dem Speichern sind zwei Lizenzierungsoptionen registriert, die auf das gleiche Lizenzierungsrecht verweisen! Der opsi-Service rechnet jede Anwendung einer der beiden Optionen auf die maximale Zahl von Installationen an, die das Lizenzierungsrecht einräumt. Deshalb liefert er in dem Fall einer Downgrade-Option für eine Einzel-PC-Lizenz (mit $maxInstallations = 1$) nur *entweder* für eine Installation von *Windows Vista* _oder für eine Installation von *Windows XP* einen Schlüssel.

14.6 Zuteilungen und Freigabe von Lizenzen

Die Anwendung einer Lizenzierungsoption für die Installation der Software auf einem Rechner führt zu einer Lizenznutzung.

Im opsi-Kontext werden Installationen skriptbasiert automatisch durchgeführt, wobei das auf den Clients abgearbeitete (Winst-) Skript Aufrufe an den zentral laufenden opsi-Service absetzt.

Im Folgenden werden die für die Lizenzverwaltung relevanten Service-Aufrufe und Skript-Befehle kurz dargestellt.

Für weitere Informationen zur Skriptsprache und zu spezifischen opsi-Kommandos s. die entsprechenden Dokumentationen, insbesondere das opsi-Winst-Handbuch.

14.6.1 opsi-Service-Aufrufe zur Anforderung und Freigabe einer Lizenz

Der opsi-Service-Befehl, mit dem z.B. das setup-Skript einer Betriebssystem-Installation eine Lizenzoption "ziehen" und den benötigten Lizenzkey vom Lizenzmanagement anfordern kann, lautet

getAndAssignSoftwareLicenseKey

Parameter sind die ID des Hosts, auf dem installiert wird und die ID des Lizenzpools, für den die Lizenz benötigt wird. Anstelle der Lizenzpool-ID kann auch eine Produkt-ID (oder eine Windows-Software-ID) als Parameter übergeben werden, falls eine entsprechende Zuordnung von Produkt bzw. Windows-Software-ID zum Lizenzpool im Lizenzmanagement registriert ist.

Analog gibt der Befehl

deleteSoftwareLicenseUsage

(wieder parametrisiert mit hostID und wahlweise Lizenzpool-ID), Product-Id oder Windows-Software-ID - eine Lizenznutzung frei und führt sie in den Pool der nicht verwendeten Lizenzierungsoptionen zurück.

Für die umfassende Dokumentation der opsi-Service-Befehle zum Lizenzmanagement s. unten.

14.6.2 Winst-Skriptbefehle für die Anforderung und Freigabe von Lizenzen

In den Winst sind die beiden client-bezogenen Befehle des Service in einen typischen Winst-Aufruf-Syntax integriert.

Ein Winst-Skript kann mit der Funktion *DemandLicenseKey* einen Schlüssel anfordern und damit die entsprechende Lizenzierungsoption "ziehen". Die Syntaxbeschreibung ist

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

Die Funktion gibt den Lizenzschlüssel (kann auch leer sein) als String-Wert zurück

```
set $mykey$ = DemandLicenseKey ("pool_office2007")
```

Der Wert kann dann für die weiteren Skriptbefehle zur Installation der Software verwendet werden.

Für die Freigabe einer Lizenzoption bzw. des Schlüssels - typischerweise in einem Winst-Deinstallationsskript benötigt - existiert der Befehl *FreeLicense* mit der analogen Syntax:

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

Die Boolesche Funktion

opsiLicenseManagementEnabled

prüft, ob das Lizenzmanagement freigeschaltet ist und kann für Skriptvariationen verwendet werden:

```
if opsiLicenseManagementEnabledDie Service-Methoden können zum Beispiel über das Kommandozeilen-
    Werkzeug
‘opsi-admin‘ aufgerufen werden.

Mit einem ’*’ gekennzeichete Parameter sind optional.

[[opsi-manual-licensemanagement-service-methods-contracts]]
==== Lizenzverträge

[source]
```

method createLicenseContract(*licenseContractId, *partner, *conclusionDate, *notificationDate, *expirationDate, *notes)

```
Die Methode erstellt einen neuen Lizenzvertragsdatensatz mit der ID
’licenseContractId’. Wird keine ’licenseContractId’ übergeben,
wird diese automatisch generiert. Bei Angabe der
’licenseContractId’ eines bestehenden Vertrages
wird dieser Vertrag entsprechend bearbeitet.

Die Parameter partner (Vertragspartner) und notes (Notizen zum Vertrag) sind frei wählbare
Die Methode gibt die licenseContractId des angelegten oder bearbeiteten Vertrags zurück.
```

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

Mit den String-Funktionen

getLastServiceErrorClass

sowie

getLastServiceErrorMessage

kann auf einen Fehler reagiert werden, wenn z.B. keine freie Lizenz mehr verfügbar ist:

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

14.6.3 Manuelle Administration der Lizenznutzung

Der opsi-Konfigurationseditor dokumentiert die über den opsi-Service registrierten Lizenzierungen auf der Tab-Seite "Lizenzenverwendung":

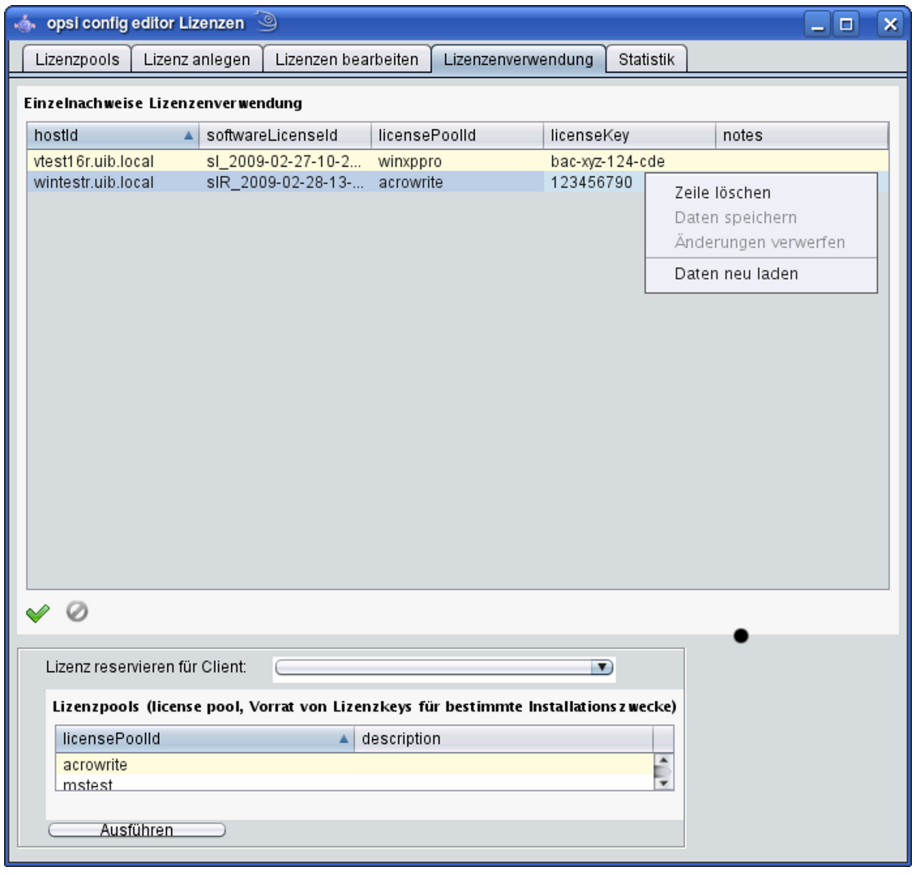


Abbildung 66: Lizenzmanagement:Tab "Lizenzenverwendung"

Die Tab-Seite ermöglicht, die Verwendung der Lizenzen auch manuell zu verwalten. Dies kann interessant sein, wenn eine Software nur vereinzelt installiert werden soll und nicht in die opsi-Verteilung eingebunden ist.

Im Einzelnen:

- Mit der Funktion "Zeilen löschen" in der Lizenzverwendungstabelle wird eine Lizenzoption wieder freigegeben.
- Der Abschnitt "Lizenz reservieren" unten auf der Seite dient dazu, eine Lizenzoption anzufordern und zu belegen.
- Durch Bearbeiten des Lizenzschlüsselfeldes in der Lizenzverwendungstabelle kann der tatsächlich für eine Lizenzierung verwendete Schlüssel (neu) bestimmt werden.

14.6.4 Erhaltung und Löschung der Lizenzenverwendungen

Wenn eine Software erneut installiert wird und der Winst mit *DemandLicenseKey* eine Lizenz anfordert, wird die vorher zugeordnete Lizenzoption weiter verwendet. Insbesondere liefert die Winst-Funktion denselben Schlüssel wie vorher.

Falls dies nicht gewünscht ist, muss die Verwendung der Lizenzierung durch den Winst mit *FreeLicense*, mit dem opsi-service-Aufruf `deleteSoftwareLicenseUsage` oder manuell aufgehoben werden.

Entsprechend bleiben bei der Reinstallation eines PCs die Lizenzverwendungen erhalten, sofern sie nicht ausdrücklich gelöscht werden. Um sie freizugeben, können auf der Tab-Seite "Lizenzenverwendung" die entsprechenden Lizenzen herausgesucht und gelöscht werden oder es kann der Serviceaufruf

deleteAllSoftwareLicenseUsages

(mit der Host-ID des betreffenden PCs als Parameter) verwendet werden.

14.7 Abgleich mit der Software-Inventarisierung

Die Tab-Seite "Abgleich mit der Inventarisierung" verzeichnet für jeden PC und jeden Lizenzpool, ob eine Lizenzpool-Verwendung mit dem opsi-Lizenzmanagement registriert ist (*used_by_opsi*) und ob auf dem PC laut Software-Inventarisierung (mittels *swaudit*) eine Windows-Software, die eine Lizenz aus dem Pool benötigen würde, installiert ist (*SWinventory_used*).

Damit die Ergebnisse von *swaudit* die faktischen Lizenzverwendungen beschreiben können, müssen die relevanten Windows-Software-IDs den jeweiligen Lizenzpools zugeordnet worden sein (Tab-Seite "Lizenzpools").

Das Lizenzmanagement zählt beim Abgleich mit der Softwareinventarisierung nur maximal 1 Vorkommen pro Client und Lizenzpool. Wenn also ein Lizenzpool *office2010* mit 10 verschiedenen Mustern aus der Softwareinventarisierung verknüpft ist, welche alle ein Hinweis darauf sind das hier *MS Office 2010* installiert ist, so wird das nur als eine Installation gezählt auch wenn alle 10 Muster auf einem Client gefunden werden.

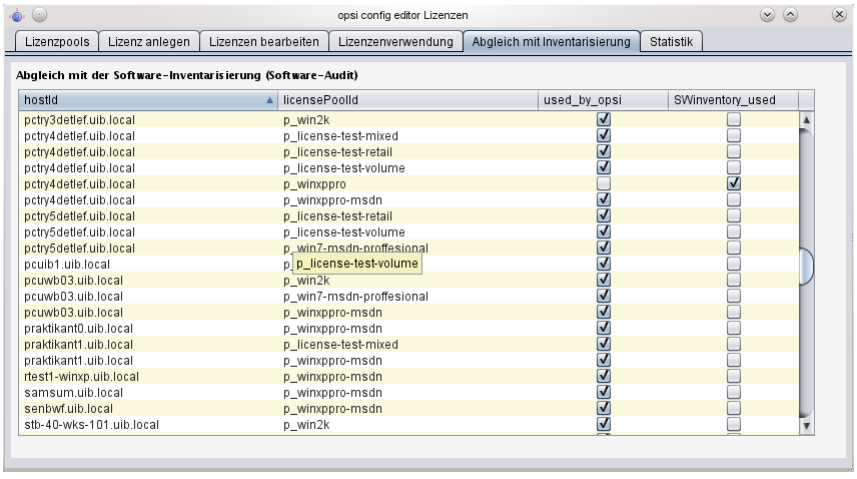


Abbildung 67: Lizenzmanagement:Tab "Abgleich mit Inventarisierung"

Die Tabelle kann wie stets per *Drag & Drop* z.B. in eine Tabellenkalkulation übernommen werden. Falls der *opsi-configed* über die entsprechenden Rechte verfügt, d.h. standalone (nicht in einer Applet-Sandbox) läuft, kann die Tabelle über eine Option des Kontextmenüs auch ausgedruckt werden.

Mittels des Configs *configed.license_inventory_extradisplayfields*, das in der Host-Parameter-Seite des Servers bearbeitet werden kann, können zusätzliche Informationen zum Client in die Tabelle aufgenommen werden.

14.8 Übersicht über den globalen Lizenzierungsstand

Die Tab-Seite "Statistik" dient dazu, eine summarische Übersicht über die genutzten und noch freien Lizenzoptionen der verschiedenen Lizenzpools zu erhalten.

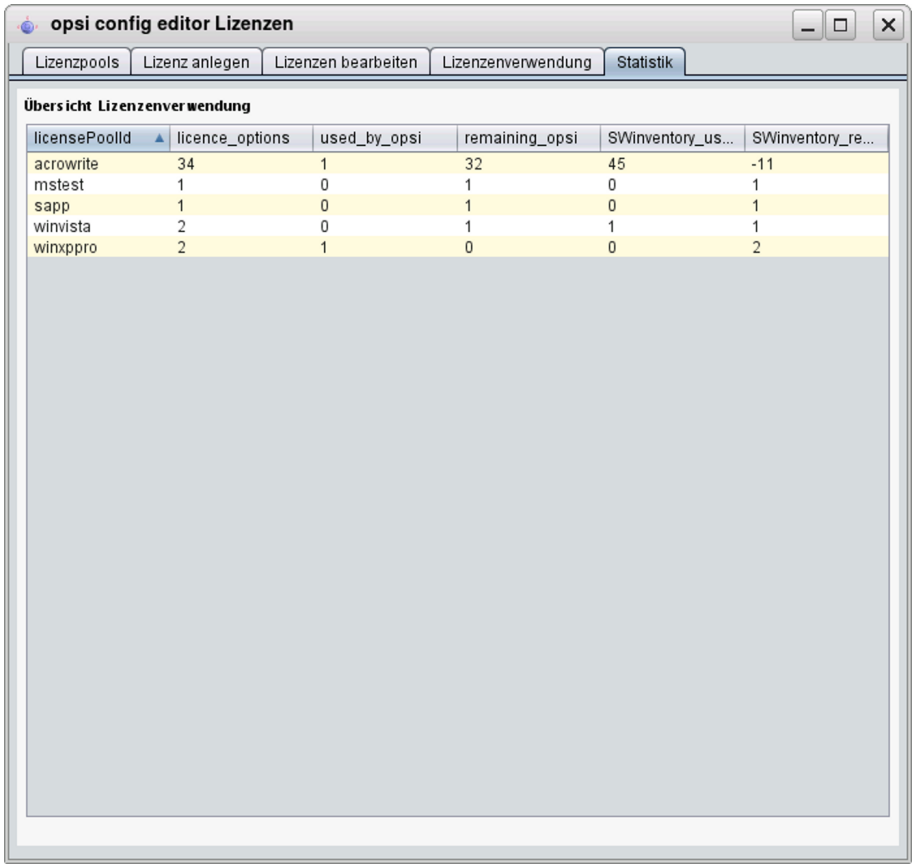


Abbildung 68: Lizenzmanagement:Tab "Statistik"

Zusätzlich zur Angabe der registrierten Lizenzverwendungen (*used by opsi*) bzw. der hiernach noch freien (*remaining...*) Lizenzen wird in die Übersicht auch die Gesamtzahl tatsächlich vorfindbarer Installationen, die eigentlich eine Lizenz benötigen, einbezogen (*SWinventory_used*)

Die Daten der Spalte *SWinventory_used* beruhen auf den Scans der Registry der Clients, die das opsi-Produkt *swaudit* durchführt, und den Zuordnungen der hiermit ermittelten Windows-Software-IDs zu den jeweiligen Lizenzpools, verwaltet in der Tab-Seite "Lizenzpools" (vgl. Abschnitt 14.3).

Über eine Option des Kontextmenüs kann die Tabelle ausgedruckt werden (aufgrund der spezifischen Security-Restriktionen nicht aus dem Applet), mit *Drag & Drop* können die Daten z.B. in eine Tabellenkalkulation übernommen werden.

14.8.1 Fall Downgrade-Option

Wenn eine Downgrade-Option konfiguriert wurde (wie in Abschnitt 14.5.1 beschrieben), äußert sich dies in der statistischen Übersicht der Lizenzverwendung wie folgt:

Eine Downgrade-Lizenz räumt je eine Lizenzierungsoption für (mindestens) zwei Lizenzpools ein. Nur eine der beiden kann tatsächlich genutzt werden. Sobald daher eine Lizenzoption gezogen ist, verringert sich in der Spalte 'remaining_opsi` in *beiden* Zeilen der Wert um je *1*. Scheinbar vermindert sich also die Zahl der verfügbaren Lizenzen um *2*! Dies spiegelt aber die tatsächliche Berechtigungssituation wider.

14.9 Service-Methoden zum Lizenzmanagement

Die Service-Methoden zum Lizenzmanagement können über das Kommandozeilenwerkzeug `opsi-admin` verwendet werden, um in einem Skript z.B. vorhandene Lizenzen aus einer Datei einzulesen.

Entsprechende Beispiele finden Sie in den Produkten *license-test-....opsi* unter http://download.uib.de/opsi4.0/-products/license-management/. Wenn Sie diese Pakete mit opsi-package-manager -i *.opsi installieren, so finden Sie unter /opt/pcbin/install/<produktname> die entsprechenden Scripte: create_license-*.sh. Hier als Beispiel das `script create_license-mixed.sh` (ziehen Sie sich im Zweifelsfall ein aktualisiertes Skript von der genannten Adresse).

```
#!/bin/bash
# This is a test and example script
# (c) uib gmbh licensed under GPL

PRODUCT_ID=license-test-mixed
# read the license key from a file
# myretailkeys.txt has one licensekey per line
MYRETAILKEYS=‘cat myretailkeys.txt‘
# myoemkeys.txt has one pair: <licensekey> <hostid.domain.tld> per line
MYOEMKEYS=‘cat myoemkeys.txt‘
# some output
echo "$PRODUCT_ID"

# this is the function to create the oem licenses
##############
createlic ()
{
while [ -n "$1" ]
do
        #echo $1
        AKTKEY=$1
        shift
        #echo $1
        AKTHOST=$1
        shift
        echo "createSoftwareLicense with oem key: ${PRODUCT_ID}-oem-${AKTKEY} for host ${AKTHOST}"
        MYLIC=‘opsi-admin -dS method createSoftwareLicense "" "c_$PRODUCT_ID" "OEM" "1" "${AKTHOST}" ""‘
        opsi-admin -d method addSoftwareLicenseToLicensePool "$MYLIC" "p_$PRODUCT_ID" "${PRODUCT_ID}-oem-${AKTKEY}"
done
}
##############

# here the script starts

# delete the existing license pool and all connected licenses
# ATTENTION: never (!) do this on a productive system
echo "deleteLicensePool p_$PRODUCT_ID"
opsi-admin -d method deleteLicensePool "p_$PRODUCT_ID" true

# delete the existing license contract
echo "deleteLicenseContract c_$PRODUCT_ID"
opsi-admin -d method deleteLicenseContract "c_$PRODUCT_ID"

# create the new license pool
# the used method has the following syntax:
```

```
# createLicensePool(*licensePoolId, *description, *productIds, *windowsSoftwareIds)
echo "createLicensePool p_$PRODUCT_ID"
opsi-admin -d method createLicensePool "p_$PRODUCT_ID" "opsi license test" \'["'$PRODUCT_ID'"]\' \'["'$PRODUCT_ID'"\
    ']\'

# create the new license contract
# the used method has the following syntax:
# createLicenseContract(*licenseContractId, *partner, *conclusionDate, *notificationDate, *expirationDate, *notes)
echo "createLicenseContract c_$PRODUCT_ID"
opsi-admin -d method createLicenseContract "c_$PRODUCT_ID" "uib gmbh" "" "" "" "test contract"

# create the new license and add the key(s)
# the used methods have the following syntax:
# createSoftwareLicense(*softwareLicenseId, *licenseContractId, *licenseType, *maxInstallations, *boundToHost, *\
    expirationDate)
# addSoftwareLicenseToLicensePool(softwareLicenseId, licensePoolId, *licenseKey)

# create the retail licenses:
for AKTKEY in $MYRETAILKEYS
do
        echo "createSoftwareLicense with retail key: ${PRODUCT_ID}-retail-${AKTKEY}"
        MYLIC=`opsi-admin -dS method createSoftwareLicense "" "c_$PRODUCT_ID" "RETAIL" "1" "" ""`
        opsi-admin -d method addSoftwareLicenseToLicensePool "$MYLIC" "p_$PRODUCT_ID" "${PRODUCT_ID}-retail-${AKTKEY}"
done

# create the oem licenses
createlic $MYOEMKEYS

# create the volume licenses
echo "createSoftwareLicense with volume key: ${PRODUCT_ID}-vol-key"
MYLIC=`opsi-admin -dS method createSoftwareLicense "" "c_$PRODUCT_ID" "VOLUME" "10" "" ""`
opsi-admin -d method addSoftwareLicenseToLicensePool "$MYLIC" "p_$PRODUCT_ID" "${PRODUCT_ID}-vol-key"#
```

14.10 Beispielprodukte und Templates

Im Downloadbereich von uib finden sich unter

http://download.uib.de/opsi4.0/products/license-management/

vier Beispielprodukte: je ein Produkt zur Verwendung von Retail, OEM und Volumenlizenzen sowie ein Produkt, welches alle drei Lizenztypen vereint.

Diese Produkte belegen bei der Installation beispielhaft Lizenzen und geben sie bei der Deinstallation wieder frei. Weiterhin hinterlassen diese Beispielprodukte auch entsprechende Spuren in der Softwareinventarisierung, die vom Lizenzmanagement zum Abgleich verwendet werden können. Alle diese Produkte enthalten (wie oben schon erwähnt) ein Shell-Skript, mit dem zu Testzwecken automatisiert die Lizenzpools, Lizenzverträge und Lizenzen angelegt werden können.

Das Standardtemplate für Winst-Skripte *opsi-template* enthält ebenfalls die notwendigen Beispiele zur Nutzung des opsi-Lizenzmanagements.

15 opsi WAN/VPN-Erweiterung

Die WAN/VPN-Erweiterung bietet opsi-Administratoren die Möglichkeit, auch Clients hinter langsamen Leitungen in opsi einzubinden. Diese Dokumentation soll die Funktionsweise dieser Erweiterung von opsi erläutern und einen Leitfaden bieten, wie man diese Erweiterung konfigurieren und pflegen kann.

15.1 Vorbedingungen für die WAN/VPN-Erweiterung

Als Erstes sei an dieser Stelle erwähnt, dass dieses Modul momentan eine kofinanzierte opsi Erweiterung ist. Weitere Details hierzu finden Sie in Abschnitt 6.

Es sind eine Reihe von Vorbedingungen nötig, um dieses Modul einsetzen zu können. Zunächst werden *Produkt-Gruppen* benötigt, diese stehen erst ab opsi 4.0 zur Verfügung. Weiterhin werden die Pakete *opsi-client-agent* und *opsi-configed* ab Version 4.0.1 benötigt.

Tabelle 2: Benötigte Pakete

opsi-Paket	**Version**
opsi-client-agent	>=4.0.1-1
opsi-winst	>=4.10.8.12
python-opsi	>=4.0.1-7
opsi-configed	>=4.0.1.6-1

15.2 Überblick über die WAN/VPN-Erweiterung

Grob betrachtet verläuft die Softwareverteilung per opsi in der Regel folgendermaßen ab:

- Der *opsi-Loginblocker* blockiert beim Systemstart die Anmeldung von Benutzern am System.
- Der *opsiclientd* nimmt Kontakt zum *opsi-configserver* auf.
- Sind *Produkt-Aktionen* für den Client gesetzt, verbindet dieser ein Netzlaufwerk mit dem *opsi-depot*.
- Der *opsi-winst* wird gestartet und nimmt ebenfalls Kontakt zum *opsi-configserver* auf.
- Der *opsi-winst* bearbeitet die gesetzten *Produkt-Aktionen*, wobei er direkt auf das Netzlaufwerk zugreift.
- Benötigte Reboots werden ausgeführt und der Prozess beginnt erneut.
- Nach dem Abschluss aller *Produkt-Aktionen* werden Log-Dateien an den *opsi-configserver* übertragen und die Anmeldung für den Anwender freigegeben.

Betrachten wir nun den Fall eines Clients in einer Außenstelle, die über eine *WAN*-Leitung an das *LAN* angebunden ist, in dem sich *opsi-configserver* und *opsi-depotserver* befinden:

- Bei der Kommunikation mit dem *opsi-configserver* werden nur geringe Datenmengen übertragen, hier tritt keine problematische Verzögerung des Softwareverteilungs-Prozesses auf.
- Das Bearbeiten der *Produkt-Aktionen*, dauert jedoch je nach Paket-Größe, Bandbreite und Latenz der *WAN*-Verbindung, sehr lange. Auch kann es bei Dateizugriffen zu Timeouts kommen.
- Der Rechner ist dementsprechend lange für den Anwender blockiert.

Sollte sich das Aufstellen eines eigenen *opsi-depotserver* in der Außenstelle nicht rentieren, kann das Problem über die Verwendung der WAN/VPN-Erweiterung gelöst werden.
Der *opsi-client-agent* kann hierbei folgendermaßen konfiguriert werden:

- Beim Systemstart findet bei einem ungefülltem Produkt-Cache keine Softwareverteilung statt. Der Login wird nicht weiter blockiert.
- Bei gesetzten *Produkt-Aktionen* beginnt der *opsiclientd* mit der Übertragung der benötigten Dateien vom *opsi-depot* auf den lokalen Rechner. Die Übertragung kann hierbei in der Bandbreite beschränkt und auch je nach aktueller Netz-Auslastung dynamisch angepasst werden.
- Nach abgeschlossener Synchronisation der Produkt-Pakete mit dem lokalen Cache wird eine Reboot-Anforderung ausgelöst.

- Der angemeldete Benutzer stimmt dem geforderten Reboot zu oder der Rechner wird später aus einem anderen Grund neu gestartet.
- Beim Systemstart wird ein gefüllter Produkt-Cache festgestellt und die Softwareverteilung findet wie gewohnt statt. Hierbei wird jedoch mit den lokalen Dateien gearbeitet. Die Installation läuft somit sogar schneller als im *LAN*.

Betrachten wir nun den Fall eines Notebooks, das in vielen Fällen beim Systemstart überhaupt keinen Kontakt zum *opsi-configserver* herstellen kann:

- Ein Kontakt-Aufbau zum *opsi-configserver* beim Systemstart läuft in den meisten Fällen in einen Timeout.
- Unter Umständen kann der Kontakt zum *opsi-configserver* erst dann hergestellt werden, wenn sich ein Benutzer am System anmeldet und über einen *VPN*-Adapter eine Verbindung zum Unternehmens-Netzwerk herstellt.
- Ohne Verbindung zum *opsi-configserver* kann keine Softwareverteilung stattfinden.

Auch dieses Problem kann über die Verwendung der WAN/VPN-Erweiterung gelöst werden.
Der *opsi-client-agent* kann hierbei folgendermaßen konfiguriert werden:

- Beim Systemstart findet bei einem ungefülltem Produkt- oder Config-Cache keine Softwareverteilung statt. Der Login wird nicht weiter blockiert.
- Bei Aktivierung eines Netzwerk-Adapters und/oder in regelmäßigen Zeitabständen wird im Hintergrund versucht, eine Verbindung zum *opsi-configserver* herzustellen.
- Ist der *opsi-configserver* erreichbar, beginnt der *opsiclientd* mit:
  - Der Synchronisation der Konfigurationen.
  - Der Übertragung der benötigten Dateien vom *opsi-depot* auf den lokalen Rechner.
    In Verbindung mit der opsi-Erweiterung *Dynamische Depot-Auswahl* findet die Datei-Übertragung immer von dem *opsi-depot* statt, zu dem die beste Netzwerkverbindung besteht.
- Nach abgeschlossener Synchronisation der Produkt-Pakete und der Konfigurationen mit dem lokalen Cache wird eine Reboot-Anforderung ausgelöst.
- Der angemeldete Benutzer stimmt dem geforderten Reboot zu oder der Rechner wird später aus einem anderen Grund neu gestartet.
- Beim Systemstart wird ein gefüllter Produkt- und Config-Cache festgestellt. Die Softwareverteilung findet wie gewohnt statt. Hierbei wird jedoch mit den lokalen Dateien und den lokalen Konfigurationen gearbeitet. Der *opsiclientd* übernimmt hierfür die Funktionen des *opsi-configservers* und des *opsi-depotservers*.
- Beim nächsten Verbindungsaufbau zum *opsi-configserver* werden die Ergebnisse (die Änderungen an den Konfigurationen, Log-Dateien ...) synchronisiert.

Der Mechanismus zur Produkt-Synchronisation kann hierbei mehrfach unterbrochen werden. Die Datei-Synchronisation setzt immer wieder am Punkt der Unterbrechung an. Bereits übertragene Daten müssen nicht erneut übertragen werden.

Da die WAN/VPN-Erweiterung die Möglichkeit eröffnet, auch Clients an opsi anzubinden, die sich außerhalb eines geschützten Firmen-Netzwerks befinden, sind zusätzliche Sicherheitsmaßnahmen bei der Kommunikation zwischen Clients und Servern zu empfehlen.
So bietet der *opsiclientd* nun die Möglichkeit, die Identität eines *opsi-servers* zu verifizieren. Hierfür wird das Keypair des SSL-Zertifikats des *opsiconfd* verwendet.
Über diesen Mechanismus können sowohl *opsi-configserver* als auch *opsi-depotserver* verifiziert werden, jedoch nur wenn die Kommunikation über den *opsiconfd* und per *SSL* erfolgt. Im Falle eines *opsi-depots* muss der Datei-Zugriff also über den *opsiconfd* per *HTTPS/WEBDAVS* erfolgen. Der Zugriff per *CIFS/SMB* wird nicht überprüft.

15.3 Caching von Produkten

Das Cachen von *Produkten* übernimmt der *ProductCacheService*, der Bestandteil des *opsiclientd* ist.
Der *ProductCacheService* synchronisiert die lokalen Kopien der in einem *Produkt* enthaltenen Dateien mit den Dateien des *Produkts* auf einem *opsi-depot*. Das Basis-Verzeichnis des Produkt-Caches ist konfigurierbar und standardmäßig auf `%SystemDrive%\opsi.org\cache\depot` gesetzt.

15.3.1 Protokoll zum Zugriff auf ein opsi-depot

Bei der Übertragung von Produkt-Dateien werden zwei Protokolle unterstützt.

- *CIFS/SMB*
- *HTTP(S)/WEBDAV(S)*

Bei der Verwendung von *CIFS/SMB* wird eine Verbindung zu der *depotRemoteUrl* hergestellt, die in den Eigenschaften eines *opsi-depots* konfiguriert ist. Im Falle von *HTTP(S)/WEBDAV(S)* wird die ebenfalls am Depot konfigurierte *depotWebdavUrl* verwendet.

Welches Protokoll verwendet wird, kann über das *Host-Parameter* `clientconfig.depot.protocol` Client-spezifisch konfiguriert werden. Die möglichen Werte sind `cifs` und `webdav`.

Anmerkung
Auch das opsi-linux-bootimage wertet diese Konfiguration aus und verwendet das angegebenen Protokoll.

15.3.2 Die .files-Datei

Basis für die Synchronisation ist die Datei `<product-id>.files`, die im Basis-Verzeichnis eines *Produkts* auf dem *opsi-depot* zu finden ist. Die Datei enthält Informationen zu allen in einem *Produkt* enthaltenen Dateien, Verzeichnissen und symbolischen Links. Jede Zeile in der Datei entspricht einer solchen Information. Die einzelnen Informations-Typen werden durch ein Leerzeichen voneinander getrennt.
Das erste Zeichen in einer Zeile gibt den Typ des Eintrags an, mögliche Werte sind:

- `d` für ein Verzeichnis
- `f` für eine Datei
- `l` für einen symbolischen Link

Abgetrennt durch ein Leerzeichen folgt der relative Pfad in einfachen Anführungszeichen.
Der nächste Eintrag entspricht der Dateigröße (bei Verzeichnissen und Links steht hier eine 0).
Im Falle einer Datei folgt noch die MD5-Summe der Datei, bei einem symbolischen Link das Ziel des Links.

Auszug einer *.files*-Datei:

```
d 'utils' 0
f 'utils/patch_config_file.py' 2506 d3007628addf6d9f688eb4c2e219dc18
l 'utils/link_to_patch_config_file.py' 0 '/utils/patch_config_file.py'
```

Die *.files*-Datei wird beim Einspielen von *Produkt-Paketen* (nach dem Lauf des postinst-Skriptes) automatisch erzeugt.

Warnung
Bei Verwendung der WAN/VPN-Erweiterung sollten die Dateien auf einem *opsi-depot* nicht manuell bearbeitet werden, da sonst die in der *.files*-Datei enthalten Informationen nicht mehr zutreffen und dies zu Fehlern bei der Synchronisation führt.

15.3.3 Ablauf des Produkt-Cachings

Die Synchronisation einer lokalen Kopie eines *Produkts* läuft folgendermaßen ab:

- Die *.files*-Datei des *Produkts* wird auf den lokalen Rechner übertragen.
- Es wird geprüft ob genügend freier Speicherplatz für das Caching vorhanden ist. Sollte der verfügbare Speicherplatz nicht ausreichen wird durch das Löschen von *Produkten* Platz geschaffen. Hierbei werden bevorzugt *Produkten* gelöscht, die seit längerem nicht mehr benötigten (synchronisiert) wurden.
- Das Cache-Verzeichnis, das die lokale Kopie enthält, wird angelegt sofern es noch nicht existiert.
- Anhand der Einträge in der *.files*-Datei werden nicht mehr benötigte Dateien und Verzeichnisse aus dem Cache-Verzeichnis entfernt.
- Die *.files*-Datei wird nun der Reihe nach durchgearbeitet.
  - Ein fehlendes Verzeichnis wird angelegt.
  - Eine fehlende Datei wird übertragen.
  - Vorhandene Dateien werden anhand der Größe und MD5-Summe überprüft und bei Abweichungen (teilweise) neu übertragen.

Das Ergebnis der Synchronisation ist eine exakte Kopie des Produkt-Verzeichnisses auf dem *opsi-depot*.

Anmerkung
Unter Windows werden keine Symbolischen Links erzeugt, statt eines Links wird eine Kopie des Link-Ziels angelegt.

Ein erfolgreich abgeschlossenes Produkt-Caching hat zur Folge, dass:

- Der Zustand `products_cached` den Wert `true` annimmt und dieser auch über einen Neustart hinweg erhalten bleibt (siehe: Abschnitt 7.3.6).
- Ein Event vom Typ `sync completed` ausgelöst wird.

15.3.4 Konfiguration des Produkt-Cachings

Die allgemeine Konfiguration des Produkt-Cachings wird in der `opsiclientd.conf` innerhalb der Sektion `[cache_service]` vorgenommen.

- `product_cache_max_size` (integer): Die maximale Größe des Produkt-Caches in Bytes. Hiermit wird sichergestellt, dass der durch das Produkt-Caching belegte Speicherplatz die konfigurierte Größe nicht überschreitet.
- `storage_dir` (string): Der Pfad zum Verzeichnis, in dem das Basis-Verzeichnis `depot` für das Produkt-Caching angelegt wird.

Weitere Konfigurationen erfolgen Event-spezifisch.
Innerhalb einer Event-Konfigurations-Sektion `[event_<event-config-id>]` existieren folgende Optionen:

- `cache_products` (boolean): Steht der Wert dieser Option auf `true` beginnt der *ProductCacheService* beim Auftreten des Events mit dem Cachen von *Produkten*, für die eine *Produkt-Aktion* gesetzt ist. Ist zusätzlich der Wert der Option `use_cached_products` auf `true` gesetzt, wird die weitere Bearbeitung des Events solange verzögert, bis das Cachen der *Produkte* abgeschlossen ist.
- `cache_max_bandwidth` (integer): Die maximale Bandbreite in Byte/s, die beim Cachen verwendet werden soll. Bei einem Wert kleiner oder gleich 0 wird keine Bandbreiten-Begrenzung vorgenommen.

- `cache_dynamic_bandwidth` (boolean): Steht der Wert dieser Option auf `true`, wird die für die Übertragung verwendete Bandbreite dynamisch angepasst. Hierbei wird der Netzwerk-Verkehr auf der Netzwerkschnittstelle zum *opsi-depot* kontinuierlich überwacht. Wird dabei Netzwerk-Verkehr festgestellt, der nicht durch das Produkt-Caching entsteht, wird die Bandbreite der Übertragung stark reduziert, um andere Anwendungen möglichst wenig zu beeinflussen. Ist die Bandbreite dynamisch reduziert und der Netzwerk-Verkehr im Wesentlichen auf das Produkt-Caching zurückzuführen, wird die dynamische Begrenzung wieder aufgehoben. Der Wert von `cache_max_bandwidth` wird auch bei Verwendung der dynamischen Bandbreiten-Begrenzung weiterhin berücksichtigt.
- `use_cached_products` (boolean): Ist dieser Wert auf `true` gesetzt, wird beim Bearbeiten der *Produkt-Aktionen* der lokale Produkt-Cache verwendet. Ist das Caching der *Produkte* zu diesem Zeitpunkt noch nicht abgeschlossen, wird die Bearbeitung des Events mit einem Fehler beendet.

15.4 Caching von Konfigurationen

Das Cachen von Konfigurationen übernimmt der *ConfigCacheService*, der Bestandteil des *opsiclientd* ist.
Der *ConfigCacheService* synchronisiert ein lokales *Client-Cache-Backend* mit dem *Config-Backend* des *opsi-configservers*.
Der *opsiclientd* bietet per *WebService* einen Zugriff auf das Backend und stellt somit eine ähnliche Funktionalität wie der *opsiconfd* bereit.

15.4.1 Das lokale *Client-Cache-Backend*

Das lokale *Client-Cache-Backend* basiert auf *SQLite* und besteht im Wesentlichen aus einer Arbeitskopie, einem Snapshot und einem Modification-Tracker, der über Änderungen an der Arbeitskopie Buch führt.
Das Basis-Verzeichnis des Config-Caches ist konfigurierbar und standardmäßig auf `%SystemDrive%\opsi.org\cache\config` gesetzt. Der Snapshot entspricht dem Stand der Konfigurationen auf dem *opsi-configserver* zum Zeitpunkt der letzten Synchronisation.
Die Arbeitskopie entspricht zu Beginn dem Snapshot und wird im Laufe der Aktionen modifiziert.

15.4.2 Ablauf der Synchronisation von Konfigurationen

Die Synchronisation der lokalen Änderungen im *Client-Cache-Backend* mit dem *Config-Backend* des *opsi-configservers* läuft folgendermaßen ab:

- Die im Modification-Tracker registrierten Änderungen an der Arbeitskopie werden auf den *opsi-configserver* übertragen. Änderungen an den Konfigurationen auf dem *opsi-configserver* seit der letzten Synchronisation werden durch Vergleich mit dem Snapshot erkannt. Kommt es bei der Rückübertragung der Modifikationen zu Konflikten greifen folgende Regeln:
  - Im Fall von Inventarisierungsdaten besitzen die Daten des Clients Priorität
  - Bei *Action-Requests* gilt der Wert des *opsi-configservers*
  - Im Fall von *Installations-Status* und *Aktions-Ergebnis* wir der Client-Wert bevorzugt.
  - Die verwendeten Software-Lizenzen werden vom Client vorgegeben. Bei der Replikation reservierte, ungenutzte Lizenzen werden wieder freigegeben.
  - Der *opsi-configserver* behält beim Zustand von *Host-Parametern* und *Product-Properties* recht.
- Der Modification-Tracker wird geleert.
- Die Log-Dateien werden übertragen.

Die Replikation des *Config-Backend* des *opsi-configservers* in das *Client-Cache-Backend* läuft folgendermaßen ab:

- Die Replikation findet nur statt, wenn auf dem *opsi-configserver Action-Requests* gesetzt sind, die *Produkt-Aktion* `always` gilt hierbei als nicht gesetzt. Ist der Zustand der *Action-Requests* seit dem letzten Replikations-Lauf unverändert, findet ebenfalls keine Replikation statt.

- Der Modification-Tracker die Arbeitskopie und der Snapshot werden geleert.
- Die zum autarken Arbeiten benötigten Konfigurationen werden repliziert.
- Sind *Action-Requests* für *Produkte* gesetzt die als lizenzpflichtig markiert wurden, wird eine Software-Lizenz aus einem, dem *Produkt* zugeordneten, *Lizenz-Pool* reserviert.
- Zusätzlich benötigte Daten, wie `auditHardwareConfig` und `modules` werden übertragen.
- Der Snapshot und die Arbeitskopie werden auf den gleichen Stand gebracht.

Eine erfolgreiche Replikation vom Server zum Client hat zur Folge, dass:

- Der Zustand `config_cached` den Wert `true` annimmt und dieser auch über einen Neustart hinweg erhalten bleibt (siehe: Abschnitt 7.3.6).
- Ein Event vom Typ `sync completed` ausgelöst wird.

15.4.3 Konfiguration des Config-Cachings

Die Konfiguration des Config-Cachings erfolgt hauptsächlich Event-spezifisch.
Innerhalb einer Event-Konfigurations-Sektion `[event_<event-config-id>]` existieren folgende Optionen:

- `sync_config_to_server` (boolean): Steht der Wert dieser Option auf `true`, beginnt der *ConfigCacheService* beim Auftreten des Events die im Modification-Tracker registrierten Änderungen zum *opsi-configserver* zu übertragen. Das Ergebnis dieser Aktion wird in jedem Fall abgewartet.
- `sync_config_from_server` (boolean): Ist dieser Wert auf `true` gesetzt, beginnt der *ConfigCacheService* mit der Replikation. Ist zusätzlich der Wert der Option `use_cached_config` auf `true` gesetzt wird die weitere Bearbeitung des Events solange verzögert, bis die Replikation abgeschlossen ist.
- `use_cached_config` (boolean): Steht der Wert dieser Option auf `true`, wird beim Bearbeiten der *Produkt-Aktionen* das *Client-Cache-Backend* verwendet. Ist die Synchronisation zu diesem Zeitpunkt noch nicht abgeschlossen wird die Bearbeitung des Events mit einem Fehler beendet.
- `post_sync_config_to_server` (boolean): Entspricht `sync_config_to_server`, wird jedoch nach Abschluss der *Produkt-Aktionen* ausgewertet.
- `post_sync_config_from_server` (boolean): Entspricht `sync_config_from_server`, wird jedoch nach Abschluss der *Produkt-Aktionen* ausgewertet.

15.5 Empfohlene Konfiguration bei Verwendung der WAN/VPN-Erweiterung

Das *opsi-client-agent*-Paket bringt eine, für die WAN/VPN-Erweiterung, vorbereitete `opsiclientd.conf` mit.
Um die WAN/VPN-Erweiterung zu aktivieren ist es lediglich notwendig, einige Events zu aktivieren und andere zu deaktivieren.
Da die Konfiguration des *opsi-client-agents* auch zentral über den Webservice erfolgen kann (siehe: Abschnitt 7.3.6), ist zu empfehlen die folgenden *Host-Parameter* anzulegen.

- `opsiclientd.event_gui_startup.active` (boolean, default: `true`)
- `opsiclientd.event_gui_startup{user_logged_in}.active` (boolean, default: `true`)
- `opsiclientd.event_net_connection.active` (boolean, default: `false`)
- `opsiclientd.event_timer.active` (boolean, default: `false`)

Über diese *Host-Parameter* können dann Events Client-spezifisch aktiviert bzw. deaktiviert werden. Die *Host-Parameter* können über den *opsi-configed* oder *opsi-admin* angelegt werden.

Zum Anlegen der *Host-Parameter* über *opsi-admin* sind die folgenden Befehle auf dem *opsi-configserver* auszuführen:

```
opsi-admin -d method config_createBool opsiclientd.event_gui_startup.active "gui_startup active" true
opsi-admin -d method config_createBool opsiclientd.event_gui_startup{user_logged_in}.active "gui_startup{user_logged_in\
    } active" true
opsi-admin -d method config_createBool opsiclientd.event_net_connection.active "event_net_connection active" false
opsi-admin -d method config_createBool opsiclientd.event_timer.active "event_timer active" false
```

Die gesetzten Standard-Werte entsprechen hierbei den Standard-Werten der mitgelieferten `opsiclientd.conf`.

Für einen WAN/VPN-Client, der Konfigurationen und Produkte cachen soll, werden die *Host-Parameter* wie folgt konfiguriert:

- `opsiclientd.event_gui_startup.active`: `false`
- `opsiclientd.event_gui_startup{user_logged_in}.active`: `false`
- `opsiclientd.event_net_connection.active`: `true`
- `opsiclientd.event_timer.active`: `true`

Die Client-spezifischen *Host-Parameter* können über den *opsi-configed* oder *opsi-admin* gesetzt werden.

Zum Setzen der *Host-Parameter* über *opsi-admin* sind die folgenden Befehle auf dem *opsi-configserver* auszuführen (im Beispiel für einen Client mit der opsi-host-Id `vpnclient.domain.de`):

```
opsi-admin -d method configState_create opsiclientd.event_gui_startup.active vpnclient.domain.de false
opsi-admin -d method configState_create opsiclientd.event_gui_startup{user_logged_in}.active vpnclient.domain.de false
opsi-admin -d method configState_create opsiclientd.event_net_connection.active vpnclient.domain.de true
opsi-admin -d method configState_create opsiclientd.event_timer.active vpnclient.domain.de true
```

Diese Konfiguration hat zur Folge, dass:

- Beim Start des Rechners kein Verbindungsaufbau zum *opsi-configserver* stattfindet.
- Beim Aktivieren einer beliebigen Netzwerk-Schnittstelle ein Verbindungsaufbau zum *opsi-configserver* versucht und mit der Synchronisation im Hintergrund begonnen wird.
- Eim `timer`-Event aktiviert wird, dass in regelmäßigen Abständen aktiv wird und ebenso einen Synchronisation-Versuch unternimmt.

15.5.1 Wahl des Protokolls für das Caching der *Produkte*

Das Caching der *Produkte* kann über die Protokolle *HTTPS/WEBDAVS* oder *CIFS/SMB* erfolgen.

Bei Verwendung von *webdav* erfolgt der Zugriff auf das *opsi-depot* über den *opsiconfd*.

- Vorteile:
  - Einfache Firewall-Konfiguration, lediglich Zugriff auf Port 4447 notwendig.
  - Prüfung des SSL-Zertifikats des *opsi-depots* möglich.
- Nachteile:
  - Der *opsiconfd* erzeugt höhere Lasten auf dem opsi-depot.

Bei Verwendung von *cifs* erfolgt der Zugriff auf das *opsi-depot* über *SAMBA*.

- Vorteile:
  - Der *SAMBA*-Server ist performant, ressourcenschonend und gut skalierbar.

- Nachteile:
  - Aufwändigere Firewall-Konfiguration, Zugriff auf SAMBA-Ports notwendig.
  - Prüfung des SSL-Zertifikats des *opsi-depots* nicht möglich.

Eine Anleitung zur Konfiguration des Protokolls finden sich im Kapitel Abschnitt 15.3.1.

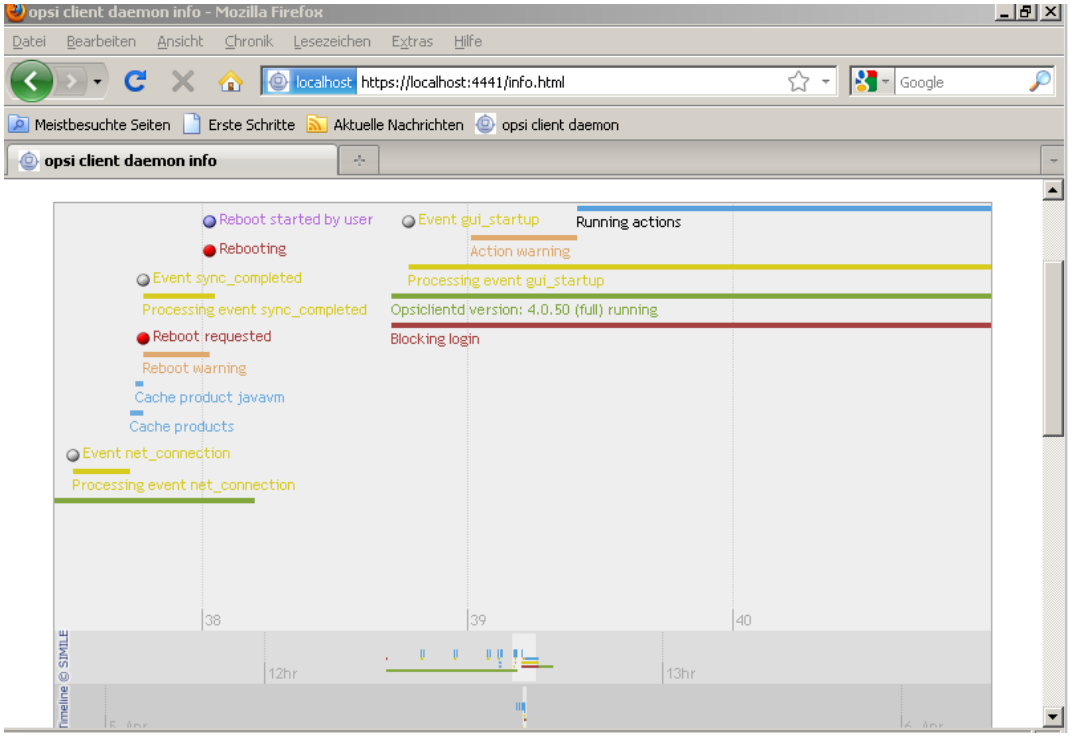


Abbildung 69: Ablauf einer Installation mit der WAN-Erweiterung in der opsiclientd-infopage

15.5.2 Prüfung der Server-Zertifikate

Um die Prüfung von SSL-Zertifikaten zu aktivieren, ist in der `opsiclientd.conf` innerhalb der Sektion `[global]` die Option `verify_server_cert` auf `true` zu setzen. Dies hat zur Folge, dass bei einem Verbindungsaufbau zu einem `opsiconfd` der opsi-server anhand des SSL-Zertifikats überprüft wird. Die Server-Zertifikate werden auf dem Client im Verzeichnis `c:\opsi.org\opsiclientd\server-certs` abgelegt. Der Dateiname des Zertifikats setzt sich aus der Server-Adresse (IP oder Name) und der Dateiendung `.pem` zusammen. Sollte beim Verbindungsaufbau kein gespeichertes Zertifikat gefunden werden, findet keine Überprüfung statt.

Tipp
Um ein geändertes Zertifikat neu zu publizieren, muss das auf den Clients vorhandene Zertifikat gelöscht werden. Hierfür steht auch die RPC-Methode `deleteServerCerts` bereit, die über das Control-Interface des opsiclientd aufgerufen werden kann.

16 opsi-server mit mehreren Depots

16.1 Konzept

Die Unterstützung von mehreren Depots in opsi hat folgende Merkmale:

- Zentrale Speicherung und Administration der Konfigurationsdaten
- Dezentrale Bereitstellung der Softwaredepots Automatisierte Verteilung der installierten Softwarepakete auf die dezentralen Depots
- Verwaltung der Clients Standortübergreifend in einem Administrationsinterface

Zur Umsetzung wurde folgendes Konzept verwirklicht:

- Die Konfigurationsdaten für alle Clients werden auf einem opsi-server (config-server) gehalten.
- Alle Clients verbinden sich über den opsi-Webservice mit dem config-server und erhalten von dort ihre Konfigurationsinformationen.

*Die Softwaredepots liegen auf dezentralen depot-servern und werden dem zentralen config-server als Netzwerkmounts zur Installation von Paketen zur Verfügung gestellt.

- Die Funktionalität zum Start von Bootimages mittels PXE wird ebenfalls auf dem dezentralen depot-server installiert. Diese wird aber zentral gesteuert.
- opsi-packet-manager: Programm zur Unterstützung von mehreren Depotshares beim Installieren und Deinstallieren von opsi-Paketen.
- Transport der opsi-Pakete via webdav auf die depot-server und Installation durch den opsiconfd via webservice-call
- Unterstützung von mehreren Depotshares im Administrationswerkzeug opsi-configed.
- Automatisierte Erkennung von Inkonsistenzen zwischen dem Master-Depotshare und anderen Depotshares anhand der hinterlegten opsi-controlfiles.
- Ermöglichung der Selektion einzelner oder mehrerer Depotshares zur Auswahl der Clients im opsi-configed.
- Unterbinden der gemeinsamen Bearbeitung von Clients, die an Depotshares hängen und zueinander inkonsistent sind.
- Zuordnung der Clients zu Depotshares über den opsi-configed, Umzug von Clients.
- Konfigurations- und Verbindungsdaten der einzelnen Depotshares über den opsi-configed editierbar machen.

Das folgende Schemata geben eine Überblick über die Kommunikation zwischen den Komponenten bei einer Situation mit einem Standort und der Situation mit einem Depotserver.

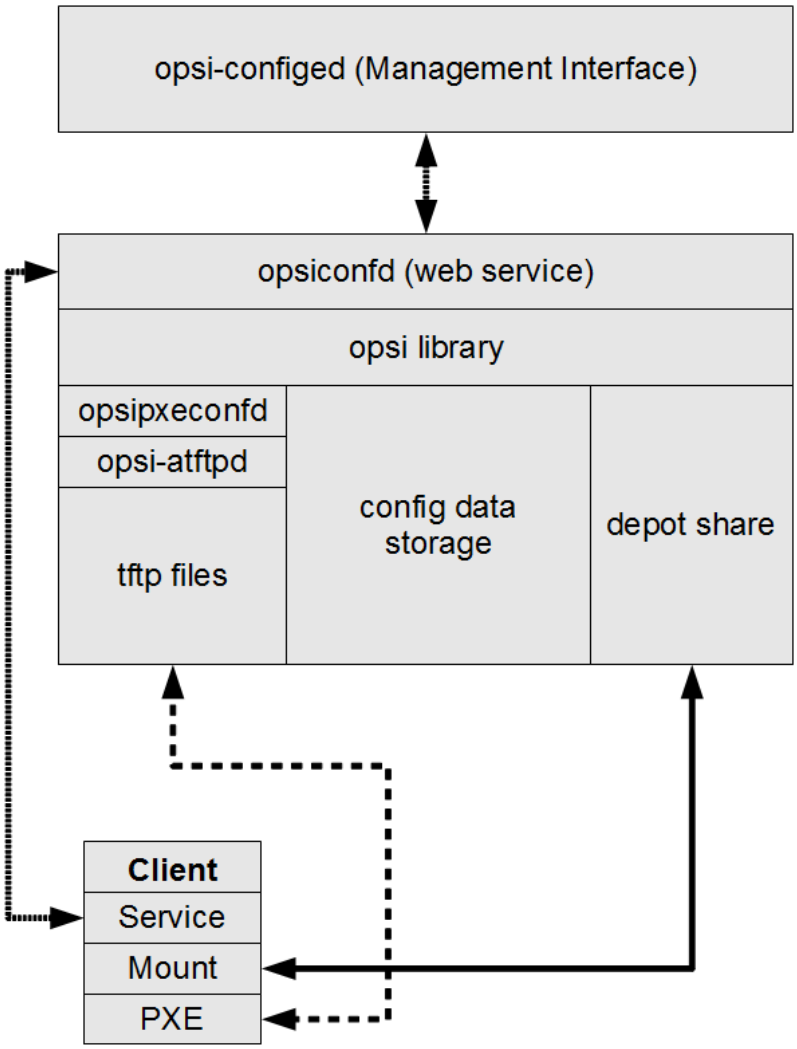


Abbildung 70: Schema: Kommunikation zwischen opsi-client und opsi-server (ein Standort)

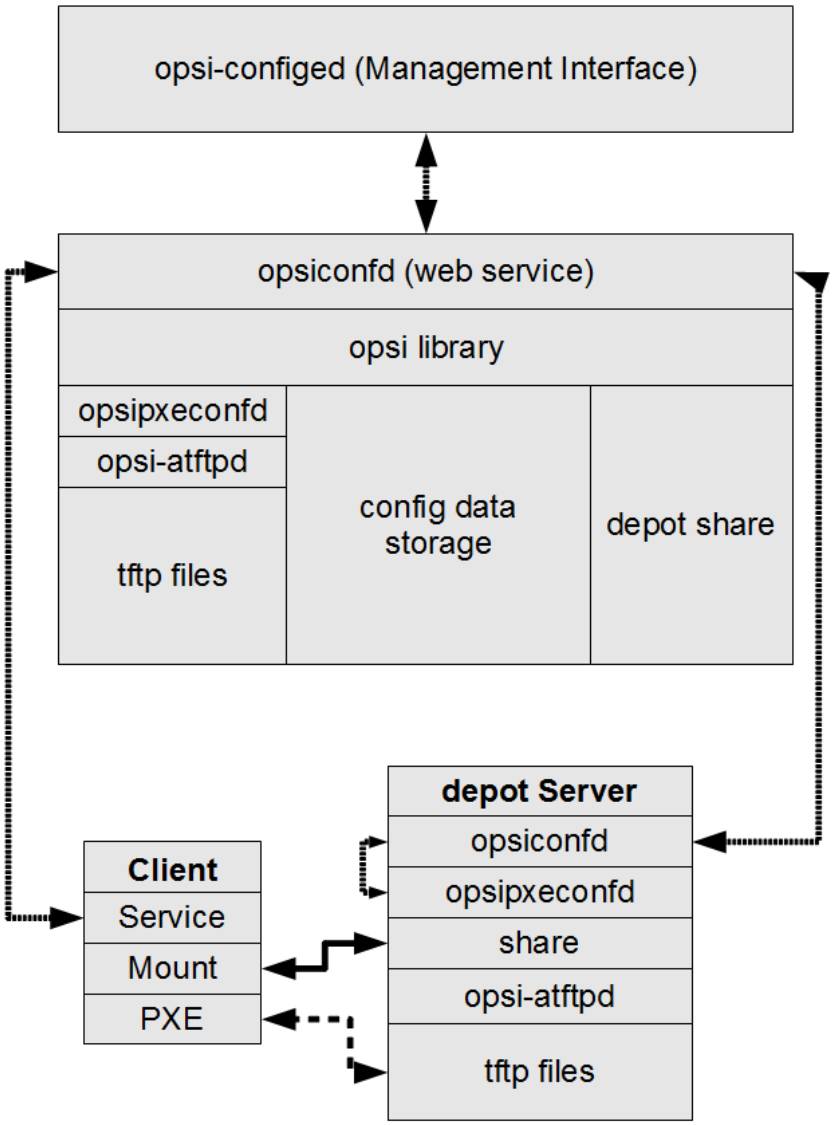


Abbildung 71: Schema: Kommunikation zwischen opsi-client und opsi-servern (mehrere Standorte)

16.2 Erstellung und Konfiguration eines (slave) depot-servers

Zur Erstellung eines externen *opsi-depotservers* wird zunächst ein normaler *opsi-server* aufgesetzt. Dann wird auf diesem neuen *opsi-server* der Befehl `opsi-setup --register-depot` mit root Rechten ausgeführt, um ihn zum externen *opsi-depotserver* zu konfigurieren. Da hierbei nicht nur der *opsi-depotserver* konfiguriert wird, sondern dieser auch noch per Webservice dem zentralen *opsi-configserver* bekannt gemacht wird, müssen username und password eines Mitgliedes der Gruppe *opsiadmin* eingegeben werden.

Beispiel:
svmdepotde.svm.local wird als *opsi-depotserver* für den *opsi-configserver sepiella.svm.local* eingerichtet:

```
svmdepotde:~# opsi-setup --register-depot
```

Nun erscheint die Maske zu dem *opsi-configserver* an dem sich dieser Server als *opsi-depotserver* anmelden soll. Diese Anmeldung muss mit einem user autorisiert werden der auf dem *opsi-configserver* Mitglied in der Gruppe *opsiadmin* ist.

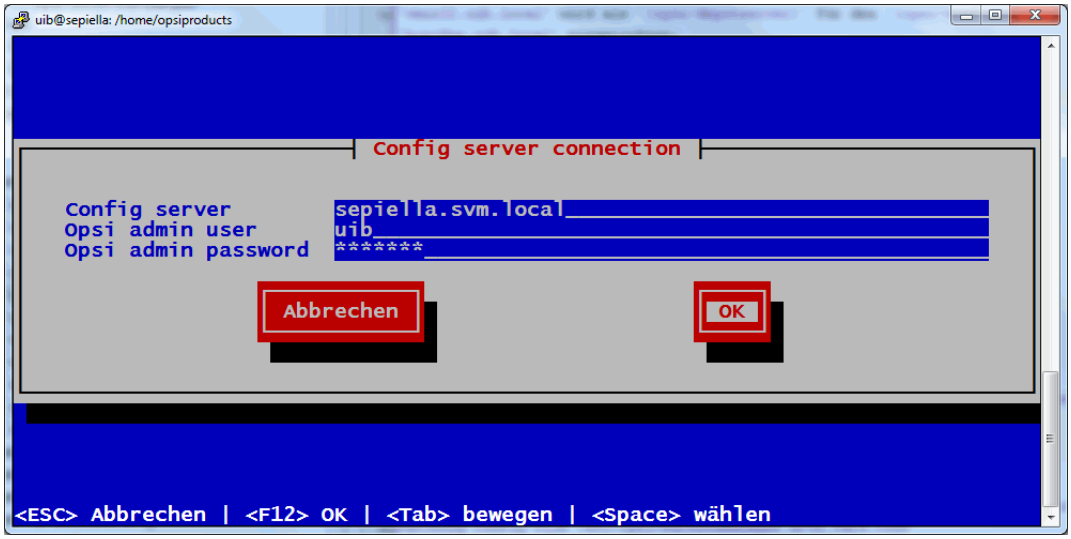


Abbildung 72: opsi-setup --register-depot : Maske Eingabe opsiadmin Account für *opsi-configserver*

Nun erscheint die Maske der Depotserver Settings.

Im Normalfall müssen Sie hier nichts ändern.

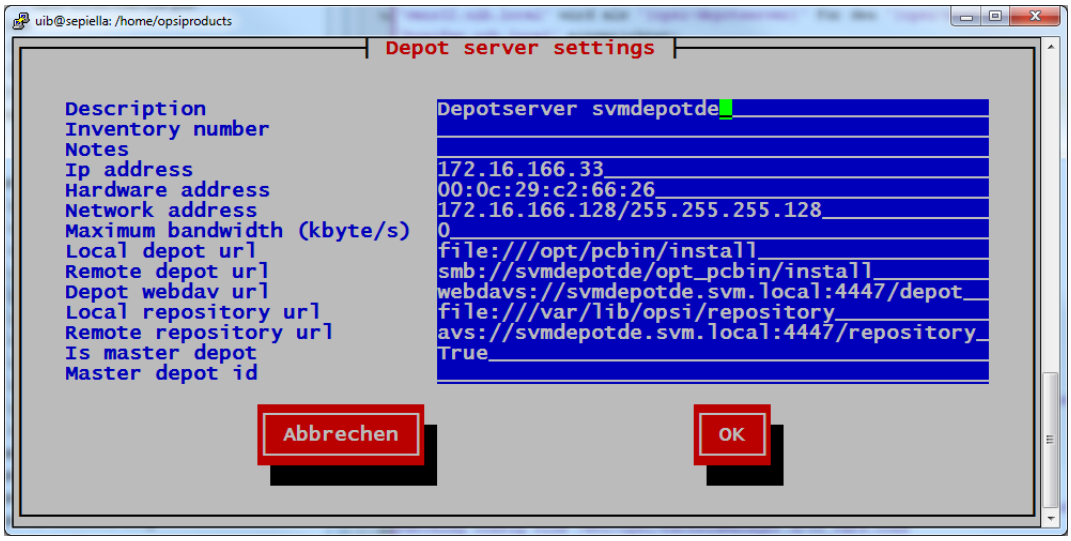


Abbildung 73: opsi-setup --register-depot : Maske Depot Settings

Nach dieser Eingabe der Daten erfolgt die eigentliche Konfiguration:

```
[5] [Apr 06 12:32:19] Getting current system config (opsi-setup|70)
[5] [Apr 06 12:32:19] System information: (opsi-setup|117)
[5] [Apr 06 12:32:19]    distributor  : Debian (opsi-setup|118)
[5] [Apr 06 12:32:19]    distribution : Debian GNU/Linux 5.0.8 (lenny) (opsi-setup|119)
[5] [Apr 06 12:32:19]    ip address   : 172.16.166.33 (opsi-setup|120)
[5] [Apr 06 12:32:19]    netmask      : 255.255.255.0 (opsi-setup|121)
[5] [Apr 06 12:32:19]    subnet       : 172.16.166.0 (opsi-setup|122)
```

```
[5] [Apr 06 12:32:19]    broadcast    : 172.16.166.255 (opsi-setup|123)
[5] [Apr 06 12:32:19]    fqdn         : svmdepotde.svm.local (opsi-setup|124)
[5] [Apr 06 12:32:19]    hostname     : svmdepotde (opsi-setup|125)
[5] [Apr 06 12:32:19]    domain       : svm.local (opsi-setup|126)
[5] [Apr 06 12:32:19]    win domain   : OPSI (opsi-setup|127)
[5] [Apr 06 12:46:03] Creating depot 'svmdepotde.svm.local' (opsi-setup|2342)
[5] [Apr 06 12:46:03] Getting depot 'svmdepotde.svm.local' (opsi-setup|2345)
[5] [Apr 06 12:46:03] Testing connection to config server as user 'svmdepotde.svm.local' (opsi-setup|2354)
[5] [Apr 06 12:46:04] Successfully connected to config server as user 'svmdepotde.svm.local' (opsi-setup|2359)
[5] [Apr 06 12:46:04] Updating backend config '/etc/opsi/backends/jsonrpc.conf' (opsi-setup|2361)
[5] [Apr 06 12:46:04] Backend config '/etc/opsi/backends/jsonrpc.conf' updated (opsi-setup|2373)
[5] [Apr 06 12:46:04] Updating dispatch config '/etc/opsi/backendManager/dispatch.conf' (opsi-setup|2375)
[5] [Apr 06 12:46:04] Dispatch config '/etc/opsi/backendManager/dispatch.conf' updated (opsi-setup|2388)
[5] [Apr 06 12:46:04] Setting rights (opsi-setup|410)
[5] [Apr 06 12:46:06] Setting rights on directory '/tftpboot/linux' (opsi-setup|482)
[5] [Apr 06 12:46:06] Setting rights on directory '/home/opsiproducts' (opsi-setup|482)
[5] [Apr 06 12:46:06] Setting rights on directory '/var/log/opsi' (opsi-setup|482)
[5] [Apr 06 12:46:06] Setting rights on directory '/etc/opsi' (opsi-setup|482)
[5] [Apr 06 12:46:06] Setting rights on directory '/var/lib/opsi' (opsi-setup|482)
[5] [Apr 06 12:46:06] Setting rights on directory '/opt/pcbin/install' (opsi-setup|482)
[5] [Apr 06 12:46:27] Restarting services (opsi-setup|2392)
[5] [Apr 06 12:46:35] Configuring client user pcpatch (opsi-setup|347)
[5] [Apr 06 12:46:35]    Creating RSA private key for user pcpatch in '/var/lib/opsi/.ssh/id_rsa' (opsi-setup|361)
[5] [Apr 06 12:46:35] Setting rights (opsi-setup|410)
[5] [Apr 06 12:46:38] Setting rights on directory '/var/lib/opsi/.ssh' (opsi-setup|482)
```

16.3 Paketmanagement auf mehreren Depots

siehe auch:
Abschnitt 4.4
Abschnitt 4.5

Zur Verwaltung der Pakete auf mehreren *opsi-depotserver* kennt der `opsi-packet-manager` die Optionen `-d` bzw. `--depots` mit denen die *opsi-depotserver* angegeben werden können auf denen ein Paket installiert bzw. deinstalliert werden soll. Mit dem Schlüsselwort *ALL* kann auf alle bekannten Depots verwiesen werden. Bei einer Installation mit der Option `-d` wird das Paket zunächst in das Verzeichnis `/var/lib/opsi/repository` des *opsi-depotserver* hochgeladen und dann von dort aus installiert.

Wird `-d` nicht angegeben, so wird nur das lokale Depot behandelt und das Paket ohne upload nach `/var/lib/opsi/repository` installiert.

Beispiel:
Installiere das Paket softprod_1.0-5.opsi auf allen Depots:

```
opsi-package-manager -d ALL -i softprod_1.0-5.opsi
```

Um die Differenzen zwischen Depots angezeigt zu bekommen wird die Option `-D` (bzw. `--differences`) verwendet.

Beispiel:
Unterschiede zwischen den bekannten Depots bezüglich des Produktes mshotfix

```
opsi-package-manager -D -d ALL mshotfix
mshotfix
    vmix12.uib.local :  200804-1
    vmix13.uib.local :  200804-1
    bonifax.uib.local:  200805-2
```

17 dynamische Depotzuweisung

17.1 Einführung

Bei der Standard Multidepot Unterstützung in opsi, sind die Clients den jeweiligen Depots fest zu geordnet. Dies wird nun erweitert durch einen Mechanismus, mit dem ein Client erkennen kann, von welchem Depot er seine Software am schnellsten beziehen kann.

Eine Zuordnung gemäß IP-Nummern ist in vielen Bereichen die einfachste und passende Lösung. In anderen Netzwerktopologien reicht dies nicht aus, z.B. bei einem sternförmigen VPN-Netzwerk.

Notwendig ist also ein Mechanismus der Clientseitig dynamisch ermittelt, zu welchem Depot die beste Verbindung möglich ist. Die konkrete Implementation eines sinnvollen Algorithmus hierfür hängt wiederum sehr von der tatsächlichen Netzwerktopologie und den Wünschen des Kunden ab. Daher ist es sinnvoll diese konfigurierbar zu gestalten.

Ausgehend von der Überlegung, dass der Client sich nach den aktuellen Gegebenheiten des Netzwerks sein Depot sucht, ist sicherzustellen, dass die zur Auswahl stehenden Depots synchron, d.h. mit den selben Software-Paketen ausgestattet, sind. Da in der Praxis nicht alle Depots einer Multidepot-Umgebung immer synchron sein werden, wird die Liste der Depots, aus der sich ein Client das für ihn Geeignetste aussuchen kann, auf jene Depots beschränkt, die zum Masterdepots des Clients synchron sind. Das Masterdepot eines Clients ist das Depot, dem der Client zugewiesen ist. Damit bestimmt das Masterdepot, welche Software in welcher Version auf dem Client installiert werden kann.

Unser Konzept hierzu sieht wie folgt aus:

Auf dem opsi-configserver wird ein Client-Script hinterlegt, welches bei Bedarf auf den Client übertragen und dort interpretiert wird. Dieses Script entscheidet, welches der zur Verfügung stehenden Depots verwendet wird. Für dieses Clientscript ist definiert: die notwendige Schnittstelle mit dem das Script die Liste der zur Auswahl stehenden Server und aktuelle Client-Konfigurationen (IP-Adresse, Netzmaske, Gateway, ...) übernimmt und über die das Script dem Client das Ergebnis des Auswahlprozesses mitteilt, sowie Schnittstellen zum Logging sowie zur Anwenderinformation über den ablaufenden Prozess.

Die konkrete Implementation dieses Scriptes kann dann jederzeit an die konkrete Situation in der jeweiligen opsi-Umgebung angepasst werden.

Der aus diesem Konzept resultierende Ablauf eines Client-Connects sieht dann wie folgt aus:

1. Der Client meldet sich per Webservice beim opsi-configserver.
2. Der opsi-configserver übermittelt dem Client die Liste der zu installierenden Software.
3. Der opsi-configserver übermittelt dem Client das zentral abgelegte Script zur Auswahl des Depotservers sowie die Liste der möglichen Depots.
4. Der Client führt das Script aus und ermittelt damit das beste Depot.
5. Der Client verbindet sich mit dem ausgewählten Depotserver, um sich von dort die zu installierende Software zu holen.
6. Der Installationsstatus wird an den opsi-configserver zurückgemeldet.

17.2 Voraussetzungen

Diese Funktion setzt opsi in der Version >= 4.0 voraus.

Dieses Modul ist momentan eine kofinanzierte opsi Erweiterung. Das bedeutet die Verwendung ist nicht kostenlos. Weitere Details hierzu finden Sie in Abschnitt 6.

Diese Funktion benötigt mind. folgende Paketstände:

```
opsi-client-agent 4.0-11
python-opsi 4.0.0.18-1
opsi-configed 4.0.1.5-1
```

17.3 Konfiguration

Das Script, welches der Client verwendet um die Depotauswahl durchzuführen, ist auf dem Server in der folgenden Datei abgelegt:
`/etc/opsi/backendManager/extend.d/70_dynamic_depot.conf`

Um die dynamische Depotauswahl für einen Client zu aktivieren, muss für diesen Client folgender Host-Parameter gesetzt werden:
clientconfig.depot.dynamic = true

Dies kann über den opsi-configed im Tab Host-Parameter geschehen.

Natürlich kann dies auch auf der Kommandozeile mit dem Befehl `opsi-admin` erledigt werden (<client-id> ist hierbei durch den FQDN, z.B. client1.uib.local des Clients zu ersetzen):

```
opsi-admin -d method configState_create \
clientconfig.depot.dynamic <client-id> [True]
```

Kontrolliert werden kann die Ausführung mittels:

```
opsi-admin -d method configState_getObjects \
[] '{"configId":"clientconfig.depot.dynamic","objectId":"<client-id>"}'
```

17.4 Editieren der Depoteigenschaften

Die Eigenschaften eines Depots werden zum Teil abgefragt, wenn ein opsi-server über den Befehl `opsi-setup --register-depot` als Depot registriert wird (siehe Abschnitt 16.2).

Die Depoteigenschaften können Sie nachträglich editieren. Dies geht sowohl im opsi Management Interface als auch auf der Kommandozeile.

Abbildung 74: Aufruf der Depoteigenschaften (2. Button von links)

Die Depoteigenschaften rufen Sie über den Button *Depoteigenschaften* rechts oben im Managementinterface auf.

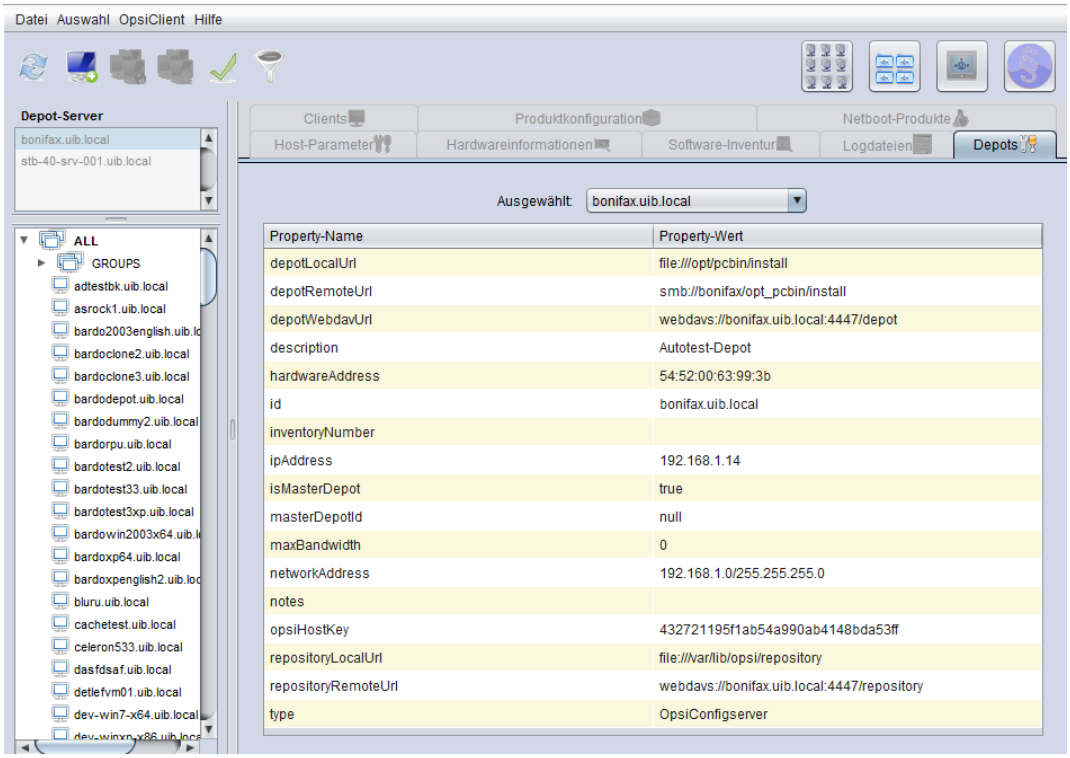


Abbildung 75: Depoteigenschaften im opsi-configed

Auf der Kommandozeile können die Depoteigenschaften ausgegeben werden mit der Methode `host_getObjects`. Hier z.B. für das Depot *dep1.uib.local*.

```
opsi-admin -d method host_getObjects [] '{"id":"dep1.uib.local"}'
```

Dieses Beispiel ergibt folgende Ausgabe:

```
[
        {
        "masterDepotId" : "masterdepot.uib.local",
        "ident" : "dep1.uib.local",
        "networkAddress" : "192.168.101.0/255.255.255.0",
        "description" : "Depot 1 an Master Depot",
        "inventoryNumber" : "",
        "ipAddress" : "192.168.105.1",
        "repositoryRemoteUrl" : "webdavs://dep1.uib.local:4447/repository",
        "depotLocalUrl" : "file:///opt/pcbin/install",
        "isMasterDepot" : true,
        "notes" : "",
        "hardwareAddress" : "52:54:00:37:c6:8b",
        "maxBandwidth" : 0,
        "repositoryLocalUrl" : "file:///var/lib/opsi/repository",
        "opsiHostKey" : "6a13da751fe76b9298f4ede127280809",
        "type" : "OpsiDepotserver",
        "id" : "dep1.uib.local",
        "depotWebdavUrl" : "webdavs://dep1.uib.local:4447/depot",
        "depotRemoteUrl" : "smb://dep1/opt_pcbin/install"
        }
]
```

Um die Depoteigenschaften auf der Kommandozeile zu editieren, wird die Ausgabe in eine Datei geschrieben:

```
opsi-admin -d method host_getObjects [] '{"id":"dep1.uib.local"}' \
> /tmp/depot_config.json
```

Die entstandene Datei (`/tmp/depot_config.json`) kann nun editiert und mit dem folgenden Befehl wieder zurückgeschrieben werden:

```
opsi-admin -d method host_createObjects < /tmp/depot_config.json
```

Die im Rahmen der dynamischen Depotzuweisung wichtigen Depoteigenschaften sind:

- *isMasterDepot*
  Muss true sein, damit diesem Depot ein Client zugewiesen werden kann.
- *networkAddress*
  Adresse des Netzwerks für den dieses Depot zuständig ist. Die Netzwerkadresse kann nach zwei Notationen angegeben werden:
  - Netzwerk/Maske Beispiel: 192.168.101.0/255.255.255.0
  - Netzwerk/Maskenbits Beispiel: 192.168.101.0/24

Ob die *networkAddress* tatsächlich zur Ermittlung des Depots ausgewertet wird, hängt natürlich von dem im Script übergebenen Algorithmus ab. Der von uib ausgelieferte Default-Algorithmus richtet sich nach diesem Kriterium.

17.5 Synchronisation der Depots

Um die Depots synchron zu halten, stellt opsi mehrere Werkzeuge bereit:

- `opsi-package-manager`
- `opsi-productupdater`

Der `opsi-package-manager` kann bei der Installation eines opsi-Paketes durch die Verwendung der Parameter `-d ALL` angewiesen werden, das Paket nicht nur auf dem aktuellen Server sondern auf allen bekannten Depots zu installieren. Beispiel:

```
opsi-package-manager -i opsi-template_1.0-20.opsi -d ALL
```

Durch die Verwendung des Parameters `-D` kann der `opsi-package-manager` angewiesen werden, die Differenzen zwischen Depots aufzulisten. Auch hierbei muss mit der Option `-d` eine Liste von Depots angegeben oder mit `-d ALL` auf alle bekannten Depots verwiesen werden. Beispiel:

```
opsi-package-manager -D -d ALL
```

Der opsi-package-manager ist also das Werkzeug, um die Synchronisation auf dem *push* Weg durchzuführen. Dahingegen ist das Werkzeug `opsi-product-updater` dafür gedacht, um Depots im *pull* Verfahren zu synchronisieren. Der `opsi-product-updater` kann dazu auf den Depots als cronjob laufen. In der Konfigurationsdatei des `opsi-product-updater` (`/etc/opsi/opsi-product-updater.conf`) ist hierzu in der Sektion *[repository_uib]* der Wert von *active* auf *false* zu setzen und in der Sektion *[repository_master]* der Wert von *active* auf *true* zu setzen. Weiterhin wird in der selben Sektion bei *opsiDepotId* die ID des Depots (FQDN) eingetragen, von dem synchronisiert werden soll. Der `opsi-product-updater` synchronisiert dann gegen die Pakete, die auf dem angegebenen Depot im Verzeichnis `/var/lib/opsi/repository` liegen.

Achtung
Wird auf einem opsi-server ein Paket mit `opsi-package-manager -i` installiert (ohne `-d`), so landet es nicht im repository Verzeichnis. Damit es dorthin kopiert wird, kann man entweder bei der Installation mit -d explizit den Namen des Depots angeben oder mit `opsi-package-manager -u <paketname>` den upload in das Repository-Verzeichnis explizit anweisen.

Bitte beachten Sie auch die Beschreibung der beiden Werkzeuge in den entsprechenden Kapiteln des opsi-Handbuchs.

17.6 Ablauf

Ist für den Client die Verwendung der dynamischen Depotzuweisung über den Host-Parameter *clientconfig.depot.dynamic* angeschaltet, so lädt dieser über den Webservice vom Server das dort hinterlegte Script und führt es aus.

Das Script, welches der Client verwendet um die Depotauswahl durchzuführen, liegt auf dem Server in der Datei: `/etc/opsi/backendManager/extend.d/70_dynamic_depot.conf`

Der in diesem Script definierten Funktion *selectDepot* werden die folgenden Parameter übergeben:

- clientConfig
  Informationen zur aktuelle Client-Konfiguration (Hash).
  Die Keys des clientConfig-Hashes sind momentan:
  - "clientId": opsi-Host-ID des Clients (FQDN)
  - "ipAddress": IP-Adresse des Netzwerk-Schnittstelle zum configserver
  - "netmask" : Netzwerk-Maske der Netzwerk-Schnittstelle
  - "defaultGateway": Standard-Gateway
- masterDepot
  Informationen zum Masterdepot (*opsi-depotserver*-Objekt). Das Masterdepot ist das Depot, dem der Client im Managementinterface zugewiesen ist. Die Attribute des übergebenen *opsi-depotserver*-Objekts entsprechen den Attributen, wie sie von `host_getObjects` (siehe Abschnitt 17.4) ausgegeben werden.
- alternativeDepots
  Informationen zu den alternativen Depots (Liste von *opsi-depotserver*-Objekten). Die Liste der alternativen Depots bestimmt sich aus den Depots, welche bezüglich der gerade benötigten Produkte identisch zum Masterdepot sind.

Auf Basis dieser Informationen kann der Algorithmus nun ein Depot aus der Liste auswählen. Das *opsi-depotserver*-Objekt des zu verwendenden Depots muss von der Funktion zurückgegeben werden. Findet der Algorithmus kein passendes Depot aus der Liste der alternativen Depots oder ist diese leer, so sollte das Masterdepot zurückgegeben werden.

17.7 Template des Auswahlscripts

Im Templatescript sind zwei Funktionen zur Auswahl eines Depots vor implementiert.
Die Funktion `depotSelectionAlgorithmByNetworkAddress` überprüft die Netzwerkadressen der übergebenen Depots und wählt jenes Depot aus, bei dem die eigene aktuelle IP-Nummer im Netz des Depots liegt.
Die Funktion `depotSelectionAlgorithmByLatency` sendet ICMP „Echo-Request“-Pakete (ping) an die übergebenen Depots und wählt das Depot mit der niedrigsten Latenzzeit aus.
Die Funktion `getDepotSelectionAlgorithm` wird vom Client aufgerufen und gibt den Algorithmus zurück, der für die AUswahl des Depots verwendet werden soll. Ohne Änderung am Templatescript wird hier die Funktion `depotSelectionAlgorithmByNetworkAddress` zurückgegeben.

```
# -*- coding: utf-8 -*-

global depotSelectionAlgorithmByNetworkAddress
depotSelectionAlgorithmByNetworkAddress = \
'''
def selectDepot(clientConfig, masterDepot, alternativeDepots=[]):
        selectedDepot = masterDepot
        logger.info(u"Choosing depot from list of depots:")
        logger.info(u"   Master depot: %s" % masterDepot)
        for alternativeDepot in alternativeDepots:
                logger.info(u"   Alternative depot: %s" % alternativeDepot)
        if alternativeDepots:
                import socket, struct
                # Calculate bitmask of host's ipaddress
```

```
                n = clientConfig['ipAddress'].split('.')
                for i in range(4):
                        n[i] = forceInt(n[i])
                ip = (n[0] << 24) + (n[1] << 16) + (n[2] << 8) + n[3]

                depots = [ masterDepot ]
                depots.extend(alternativeDepots)
                for depot in depots:
                        if not depot.networkAddress:
                                logger.warning(u"Network address of depot '%s' not known" % depot)
                                continue
                        (network, netmask) = depot.networkAddress.split(u'/')
                        while (network.count('.') < 3):
                                network = network + u'.0'
                        if (netmask.find('.') == -1):
                                netmask = forceUnicode(socket.inet_ntoa(struct.pack('>I',0xffffffff ^ (1 << 32 - \
    forceInt(netmask)) - 1)))
                        while (netmask.count('.') < 3):
                                netmask = netmask + u'.0'

                        logger.debug(u"Testing if ip %s is part of network %s/%s" % (clientConfig['ipAddress'], network\
    , netmask))

                        n = network.split('.')
                        for i in range(4):
                                n[i] = int(n[i])
                        network = (n[0] << 24) + (n[1] << 16) + (n[2] << 8) + n[3]
                        n = netmask.split('.')
                        for i in range(4):
                                n[i] = int(n[i])
                        netmask = (n[0] << 24) + (n[1] << 16) + (n[2] << 8) + n[3]

                        wildcard = netmask ^ 0xFFFFFFFFL
                        if (wildcard | ip == wildcard | network):
                                logger.notice(u"Choosing depot with networkAddress %s for ip %s" % (depot.\
    networkAddress, clientConfig['ipAddress']))
                                selectedDepot = depot
                                break
                        else:
                                logger.info(u"IP %s does not match networkAddress %s of depot %s" % (clientConfig['\
    ipAddress'], depot.networkAddress, depot))
        return selectedDepot
'''

global depotSelectionAlgorithmByLatency
depotSelectionAlgorithmByLatency = \
'''
def selectDepot(clientConfig, masterDepot, alternativeDepots=[]):
        selectedDepot = masterDepot
        logger.info(u"Choosing depot from list of depots:")
        logger.info(u"   Master depot: %s" % masterDepot)
        for alternativeDepot in alternativeDepots:
                logger.info(u"   Alternative depot: %s" % alternativeDepot)
        if alternativeDepots:
                from OPSI.Util.Ping import ping
                from OPSI.Util.HTTP import urlsplit
                depots = [ masterDepot ]
                depots.extend(alternativeDepots)
                latency = {}
                for depot in depots:
                        if not depot.repositoryRemoteUrl:
                                continue
                        try:
                                (scheme, host, port, baseurl, username, password) = urlsplit(depot.repositoryRemoteUrl)
                                latency[depot] = ping(host)
                                logger.info(u"Latency of depot %s: %0.3f ms" % (depot, latency[depot]*1000))
                        except Exception, e:
                                logger.warning(e)
```

```
                if latency:
                        minValue = 1000
                        for (depot, value) in latency.items():
                                if (value < minValue):
                                        minValue = value
                                        selectedDepot = depot
                        logger.notice(u"Choosing depot %s with minimum latency %0.3f ms" % (selectedDepot, minValue\
    *1000))
        return selectedDepot
'''

def getDepotSelectionAlgorithm(self):
        #return depotSelectionAlgorithmByLatency
        return depotSelectionAlgorithmByNetworkAddress
```

17.8 Logging

Wenn die dynamische Depotzuweisung aktiviert ist, so finden sich entsprechende Eintragungen von der Depotauswahl im `opsiclientd.log`. Hier der Log einer gekürzten Beispielsitzung. In dieser ist der Server `bonifax.uib.local` Configserver und Masterdepot für den Client `pctrydetlef.uib.local`. Als Masterserver hat die `bonifax` hier die Netzwerkaddresse `192.168.1.0/255.255.255.0`. Als alternatives Depot steht die `stb-40-srv-001.uib.local` zur Verfügung mit der Netzwerkaddresse `192.168.2.0/255.255.255.0`. Der Client `pctry4detlef.uib.local` hat die IP-Adresse `192.168.2.109`, liegt also im Netz des alternativen Depots.

```
(...)
[6] [Dec 02 18:25:27] [ opsiclientd                   ] Connection established to: 192.168.1.14   (HTTP.pyo|421)
[5] [Dec 02 18:25:28] [ event processing gui_startup  ]    [ 1] product opsi-client-agent:   setup   (EventProcessing.\
    pyo|446)
[5] [Dec 02 18:25:28] [ event processing gui_startup  ] Start processing action requests   (EventProcessing.pyo|453)
[5] [Dec 02 18:25:28] [ event processing gui_startup  ] Selecting depot for products [u'opsi-client-agent']   (Config.\
    pyo|314)
[5] [Dec 02 18:25:28] [ event processing gui_startup  ] Selecting depot for products [u'opsi-client-agent']   (__init__\
    .pyo|36)
(...)
[6] [Dec 02 18:25:28] [ event processing gui_startup  ] Dynamic depot selection enabled   (__init__.pyo|78)
(...)
[6] [Dec 02 18:25:28] [ event processing gui_startup  ] Master depot for products [u'opsi-client-agent'] is bonifax.uib\
    .local   (__init__.pyo|106)
[6] [Dec 02 18:25:28] [ event processing gui_startup  ] Got alternative depots for products: [u'opsi-client-agent']   (\
    __init__.pyo|110)
[6] [Dec 02 18:25:28] [ event processing gui_startup  ] 1. alternative depot is stb-40-srv-001.uib.local   (__init__.\
    pyo|112)
(...)
[6] [Dec 02 18:25:28] [ event processing gui_startup  ] Verifying modules file signature   (__init__.pyo|129)
[5] [Dec 02 18:25:28] [ event processing gui_startup  ] Modules file signature verified (customer: uib GmbH)   (\
    __init__.pyo|143)
(...)
[6] [Dec 02 18:25:28] [ event processing gui_startup  ] Choosing depot from list of depots:   (<string>|4)
[6] [Dec 02 18:25:28] [ event processing gui_startup  ]    Master depot: <OpsiConfigserver id 'bonifax.uib.local'>   (<\
    string>|5)
[6] [Dec 02 18:25:28] [ event processing gui_startup  ]    Alternative depot: <OpsiDepotserver id 'stb-40-srv-001.uib.\
    local'>   (<string>|7)
[5] [Dec 02 18:25:28] [ event processing gui_startup  ] Choosing depot with networkAddress 192.168.2.0/255.255.255.0 \
    for ip 192.168.2.109   (<string>|40)
[5] [Dec 02 18:25:28] [ event processing gui_startup  ] Selected depot is: <OpsiDepotserver id 'stb-40-srv-001.uib.\
    local'>   (__init__.pyo|171)
(...)
[5] [Dec 02 18:25:28] [ event processing gui_startup  ] Mounting depot share smb://stb-40-srv-001/opt_pcbin/install   (\
    EventProcessing.pyo|415)
(...)
```

18 opsi Software On Demand (Kiosk-Mode)

18.1 Einführung

Das Software-On-Demand-Modul bietet opsi-Administratoren die Möglichkeit, ihren Anwendern eine Auswahl an Produkten zur Verfügung zu stellen. Diese Produkte können vom Anwender, ohne Eingriff von einem Administrator, ausgewählt und die Installation gestartet werden. Diese Dokumentation soll die Funktionsweise des Software-On-Demand-Moduls von opsi erläutern und einen Leitfaden bieten, wie man dieses Modul konfigurieren und pflegen kann.

18.2 Vorbedingungen für das Modul

Es sind eine Reihe von Vorbedingungen nötig, um dieses Modul einsetzen zu können. Zunächst werden Produkt-Gruppen benötigt, diese stehen erst ab opsi 4.0 zur Verfügung. Weiterhin werden die Pakete opsi-client-agent und opsi-configed ab Version 4.0.1 benötigt.

Tabelle 3: Benötigte Pakete

opsi-Paket	**Version**
opsi-client-agent	>=4.0.1-3
opsi-winst	>=4.10.8.12
python-opsi	>=4.0.1-7
opsi-depotserver	>=4.0.1-2
opsi-configed	>=4.0.1.6-1

Das Software-On-Demand Modul wurde auf folgenden Browsern getestet und freigegeben:

- Internet Explorer 8
- Mozilla Firefox 3.6.15

18.3 Konfiguration

Die Konfiguration dieses Moduls basiert auf Produktgruppen und Config-Variablen. Die verwendeten Config-Variablen sind:

- software-on-demand.active
- software-on-demand.product-group-ids
- software-on-demand.show-details

Diese Config-Variablen werden beim Einspielen des opsi-depotserver-Pakets angelegt.

Achtung
Bitte bedenken Sie die Mindestanforderung des opsi-depotserver-Pakets.
(Siehe Kapitel: Vorbedingungen für das Modul).

Sollten die Config-Variablen nicht zur Verfügung stehen, können sie über `opsi-setup` erzeugt werden:

```
opsi-setup --init-current-config
```

18.3.1 Produktgruppen pflegen

Am komfortabelsten kann man Produktgruppen mit dem opsi-configed anlegen und pflegen. Dafür wechselt man zuerst auf den Tab *Produktkonfiguration*.

Tipp
Seit Version 4.0.1.6 des *opsi-configed* kann man direkt zur Produktkonfiguration wechseln, ohne vorher einen Client auszuwählen.

Die Produktgruppen-Leiste befindet sich über der eigentlichen Produktliste.

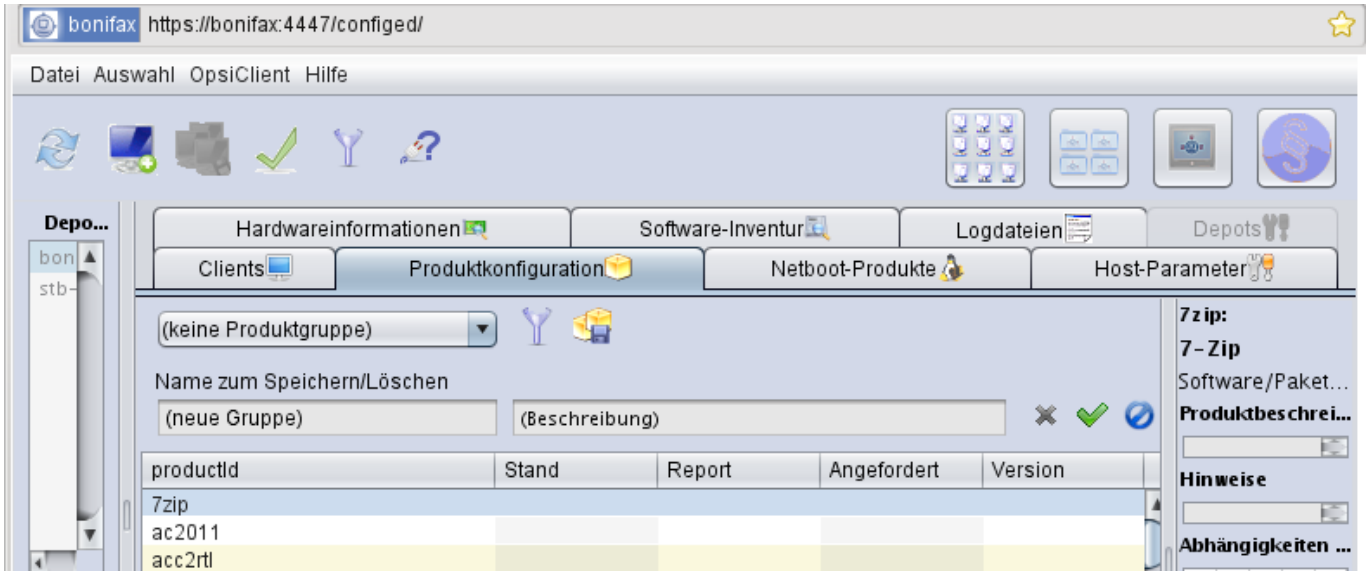


Abbildung 76: Ausschnitt von der Produktgruppen-Leiste

Mit dem Dropdown-Feld kann man Produktgruppen auswählen, um sie zu bearbeiten. Sobald eine Gruppe ausgewählt wurde, werden die dazugehörigen Produkte markiert.
Mit dem zweiten Icon kann man die Filterfunktion ein-, bzw. ausschalten. Bei aktiviertem Filter werden nur die Produkte angezeigt, die der gewählten Produktgruppe zugeordnet sind. Zur Bearbeitung von Produktgruppen aktiviert man die erweiterte Ansicht durch Klick auf das Icon "Paketgruppen mit Diskette". In dieser Ansicht kann eine neue Gruppe, optional Beschreibung, angelegt werden. Durch einen Klick auf den roten Haken, wird die neue Gruppe gespeichert.
Die Zuordnung zur Produktgruppe kann durch Selektieren bzw. Deselektieren von Produkten in der Produktliste bearbeitet werden (Die Taste `<STRG>` gedrückt halten und Produkte auswählen oder abwählen).

18.3.2 Software-On-Demand-Modul konfigurieren

Mit Hilfe der bereits erwähnten Config-Variablen kann das *SoftwareOnDemand-Modul* flexibel konfiguriert werden. Folgende Tabelle zeigt eine Übersicht der Konfigurationen:

Tabelle 4: Übersicht über die Config-Variablen des SoftwareOnDemand-Moduls

Konfiguration	Beschreibung	Mögliche Werte
software-on-demand.active	Aktiviert bzw. Deaktiviert das Modul.	true/false

Tabelle 4: (continued)

Konfiguration	Beschreibung	Mögliche Werte
software-on-demand.product-group-ids	Produktgruppen mit Produkten, die für Software-On-Demand zur Verfügung stehen sollen.	Liste von Produktgruppen
software-on-demand.show-details	Zeigt dem Anwender erweiterte Informationen an.	true/false

Es gibt zwei Möglichkeiten diese Konfigurationsobjekte zu verwenden: Systemweit oder pro Client. Die folgenden zwei Unterkapitel gehen auf die zwei verschiedenen Konfigurationsmöglichkeiten näher ein.

Systemweite Konfiguration

Die Einstellungen gelten systemweit als Vorgabe für jeden Client.

Die Konfigurationen können im *opsi-configed* im Modul Servereigenschaften im Tab Host-Parameter bearbeitet werden.

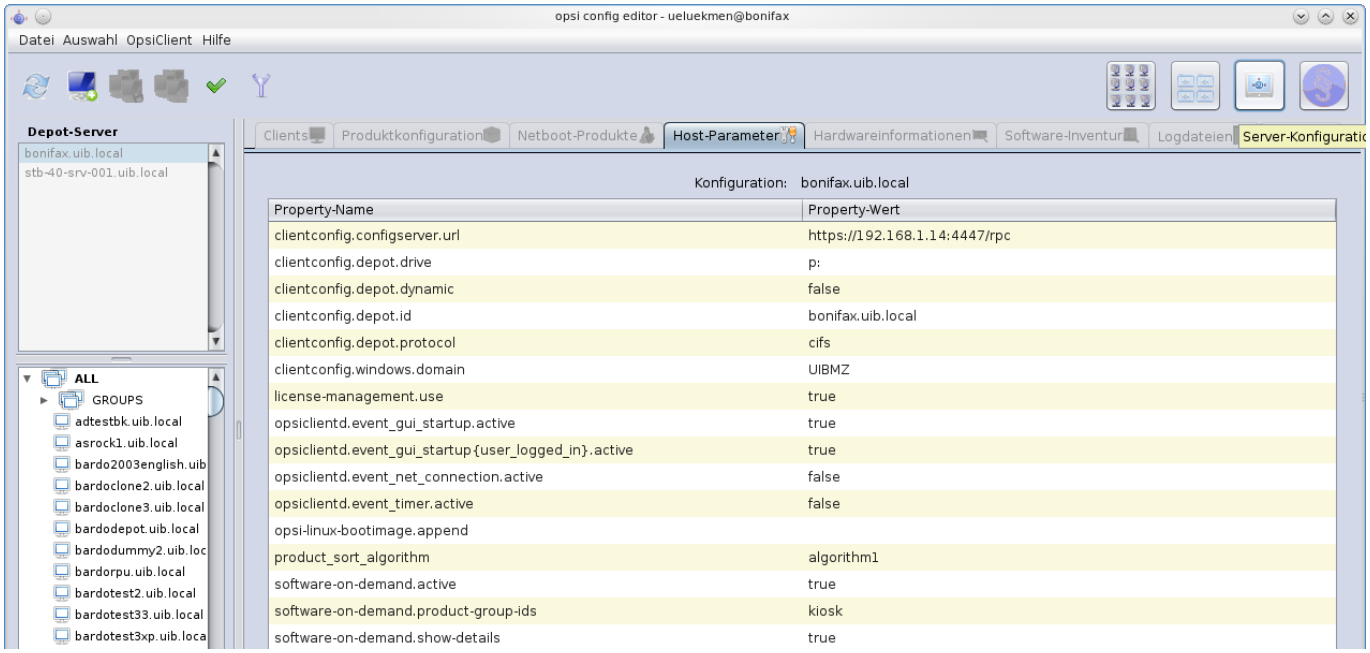


Abbildung 77: Ausschnitt von Serverkonfigurations-Modul des configed

Alternativ kann man die Konfigurationen auf dem Server mittels des folgenden Befehls anpassen:

```
opsi-setup --edit-config-defaults
```

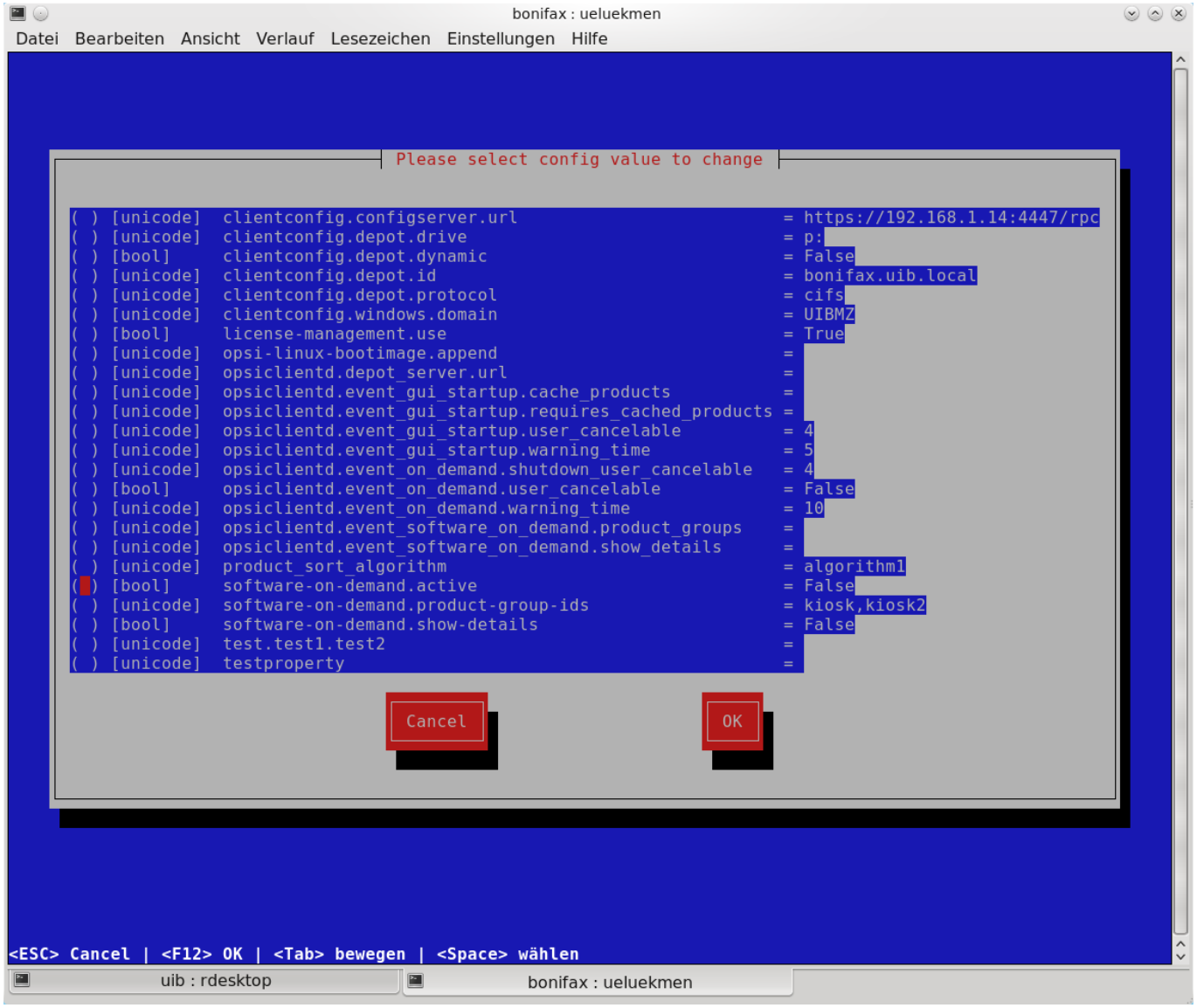


Abbildung 78: Ausschnitt von edit-config-defaults über opsi-setup

Tipp
Natürlich ist eine Bearbeitung auch über die opsi-python-API oder über `opsi-admin` möglich.

Client-spezifische Konfiguration

Die Client-spezifische Konfiguration macht dann Sinn, wenn zum Beispiel nur ein Teil der opsi-Clients Zugriff auf dieses Modul haben soll, oder wenn man verschiedenen Clients unterschiedliche Produktgruppen zur Verfügung stellen will.

Dies erreicht man durch die Konfiguration von Client-spezifischen Host-Parametern. Diese kann man wiederum auf verschiedenen Wegen bearbeiten. Die komfortabelste Möglichkeit diese Konfiguration zu bearbeiten, bietet der opsi-configed. Dafür wählt man im opsi-configed einen oder mehrere Clients (eventuell auch alle Clients einer Clientgruppe) und wechselt auf den Tab Host-Parametern.

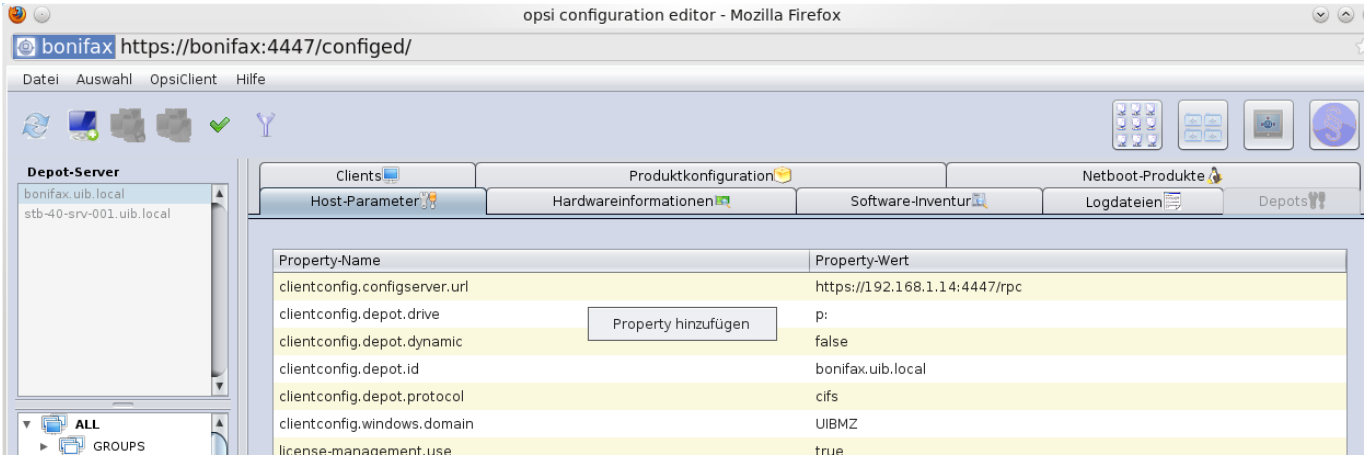


Abbildung 79: Ausschnitt von Host-Parametern

Diese Einstellungen überschreiben die systemweiten Vorgaben.

18.3.3 opsiclientd Event-Konfiguration

Beim Installieren von Produkten über das Software-On-Demand-Modul stehen dem Anwender zwei Möglichkeiten zur Verfügung, die Installation zu starten:

- beim nächsten Systemstart ausführen
- sofort ausführen

Wählt der Benutzer an dieser Stelle die Möglichkeit *beim nächsten Systemstart ausführen*, werden die Produkte nur auf *setup* gesetzt. Wird *sofort ausführen* gewählt, erzeugt der *opsiclientd* ein Event vom Typ *software on demand*. Dieses Event kann, wie jedes andere Event auch, in der `opsiclientd.conf` konfiguriert werden. In der im *opsi-client-agent* enthaltenen `opsiclientd.conf` ist bereits eine Konfiguration enthalten, die angepasst werden kann.

18.3.4 Anpassung an Corporate Identity

Das Erscheinungsbild im Browser des Software-On-Demand-Moduls kann an die firmeneigene Corporate Identity angepasst werden. Dazu muss die CSS-Datei: *opsiclientd.css* angepasst werden. Auf dem Client liegt diese Datei unter:

C:\Programme\opsi.org\opsi-client-agent\opsiclientd\static_html

Diese kann durch editieren und neu laden angepasst werden. Diese Änderung muss auf den Server kopiert werden, um bei Neuinstallationen des opsi-client-agenten die Änderungen mit zu verteilen. Dazu muss die CSS-Datei und eventuell die Logo-Datei auf den Server ins Verzeichnis:

/opt/pcbin/install/opsi-client-agent/files/opsi/dist/opsiclientd/static_html

kopiert werden. Ein nachträgliches Rechte nachziehen hilft Folgefehler zu vermeiden:

```
opsi-setup --set-rights /opt/pcbin/install/opsi-client-agent
```

Achtung
Die Änderungen werden momentan nicht gesichert und würden bei einer Neuinstallation des opsi-Pakets opsi-client-agent überschrieben werden. Bitte denken Sie daran, die Dateien vor einem Upgrade zu sichern.

18.4 Verwendung

Das Software-On-Demand-Modul stellt über den `opsiclientd` eine Webanwendung zur Verfügung. Diese ist über den jeweiligen Clients unter der URL https://localhost:4441/swondemand erreichbar.

Wenn der opsi-client-agent während der Installation merkt, dass die Konfiguration: *software-on-demand.active* auf *true* gesetzt wurde, wird automatisch während der Installation auf dem Client ein Startmenü-Eintrag erstellt, über den die Webanwendung direkt aufgerufen werden kann. Diesen findet man dann unter: *Start* → *Programme* → *opsi.org* → *software-on-demand*. Über diesen Startmenü-Eintrag wird der Standardbrowser mit der oben genannten URL aufgerufen.

Die Anzeige wird beeinflusst durch die Konfiguration vom *software-on-demand.show-details*. Durch diese Konfiguration werden entweder nur minimale bzw. viele Eigenschaften der Produkte gezeigt.

Auf das Modul kann auch über das Netzwerk zugegriffen werden, hierbei ist jedoch eine Authentifizierung notwendig.

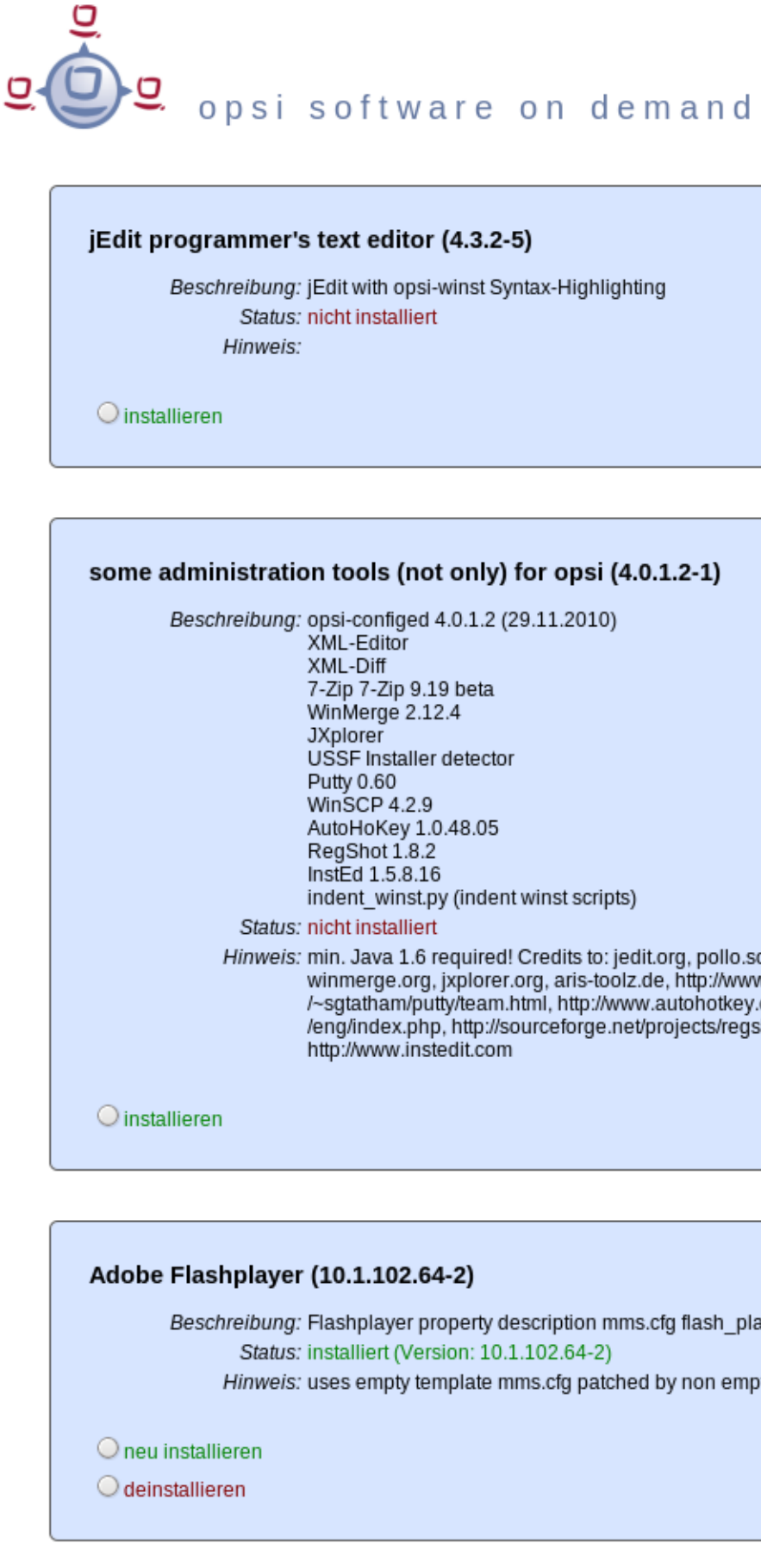


Abbildung 80: Ausschnitt von der Übersichtsseite von Software-on-demand

Aus der Liste, die angezeigt wird, kann sich der Anwender die Software aussuchen und zum Installieren auswählen; dies geschieht über die Aktivierung der Checkbox: *installieren*. Wenn die Software schon installiert war, wird *neu installieren* und zusätzlich *deinstallieren* zur Auswahl gestellt. (Abhängige Pakete, die eventuell über die Abhängigkeitssteuerung von opsi mit installiert wurden, werden bei dieser Deinstallation nicht mit deinstalliert, da in diesem Zustand nicht hundert prozentig festgestellt werden kann, ob die Abhängigkeit nur dieses Paket betrifft.)

Nach dem die Auswahl abgeschlossen wurde, kommt man durch den Button: *weiter* auf die nächste Seite.

Auf der nächsten Seite wird eine Übersicht über die anstehenden Aktionen angezeigt, auch diese Seite ist über die Konfiguration von *software-on-demand.show-details* beeinflussbar. Wenn diese Konfiguration auf *true* steht, wird neben der Auswahl des Anwenders noch zusätzlich angezeigt, welche Pakete über eine Abhängigkeit auf setup gesetzt wurden und welche Pakete schon auf setup standen.

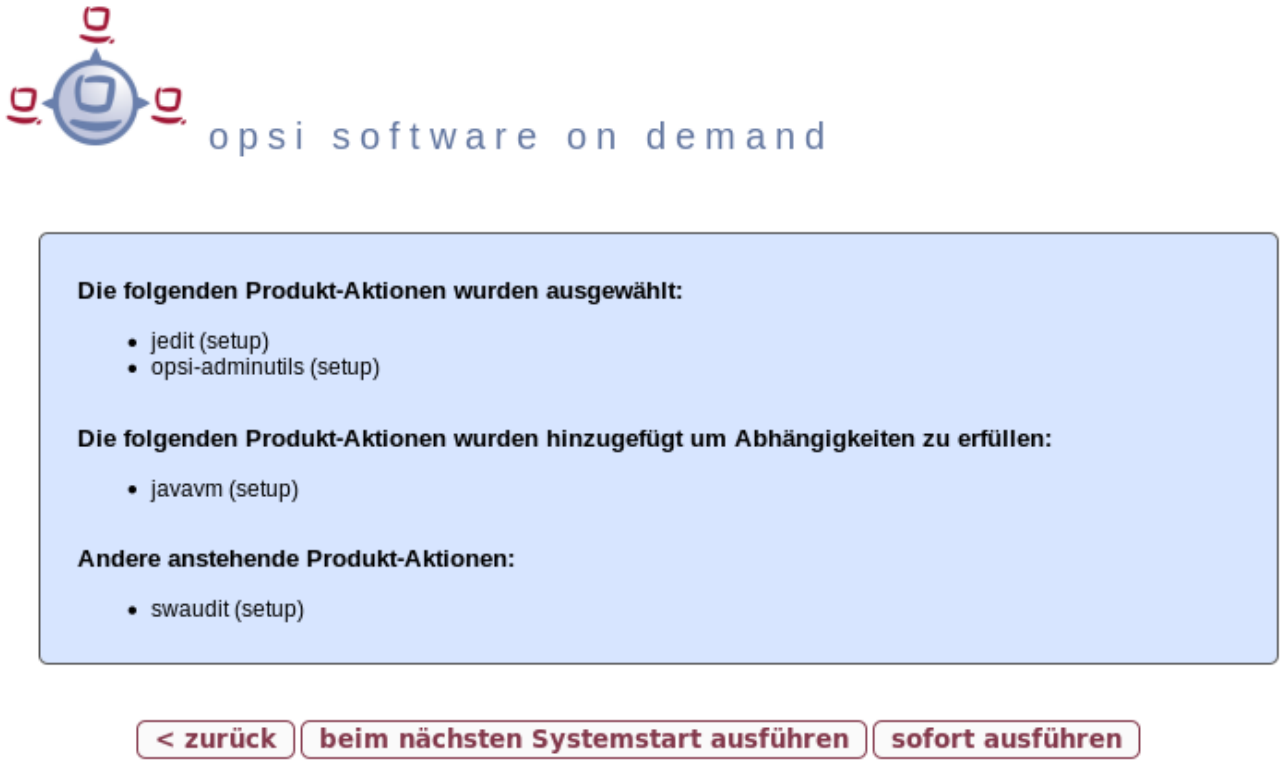


Abbildung 81: Ausschnitt von der Übersichtsseite der anstehenden Aktionen

Wie man oben im Ausschnitt erkennen kann, hat man nun drei Auswahlmöglichkeiten. Zu diesem Zeitpunkt wurden die Änderungen noch nicht an den opsi-Service übertragen. Hier hat man noch die Möglichkeit mit dem Button: *zurück* auf die Übersichtsseite zurück zu wechseln, um die Auswahl anzupassen. Der Button *beim nächsten Systemstart ausführen* schickt die Änderungen an den opsi-Service weiter und die Änderungen werden für den nächsten Systemstart vorgemerkt. Der Button *sofort ausführen* löst das oben genannte Event aus und die Installationen werden anhand der Eventkonfiguration sofort ausgeführt.

18.5 Besonderheiten

Folgende Besonderheiten gelten für das Software-On-Demand Modul:

- Abhängigkeiten werden automatisch aufgelöst

  - Software, die von Software aus der Demand-Gruppe abhängig ist, wird automatisch falls benötigt auf setup gesetzt, ohne Einfluss des Anwenders.

- Software die schon auf setup steht

  - In diesem Fall, wird die Checkbox: *installieren*, schon bei der Übersichtsseite aktiviert.

19 opsi Erweiterung *User Profile Management*

19.1 Vorbedingungen für die opsi Erweiterung *User Profile Management*

Dieses Modul ist momentan eine kofinanzierte opsi Erweiterung.
Es sind eine Reihe von Vorbedingungen nötig, um dieses Modul einsetzen zu können. Das bedeutet, das Sie zum Einsatz eine Freischaltdatei benötigen. Diese Freischaltung erhalten Sie wenn Sie die Erweiterung kaufen. Zu Evaluierungszwecken stellen wir Ihnen auch eine zeitlich befristete Freischaltung kostenlos zur Verfügung (→ mail an info@uib.de).
Weitere Details hierzu finden Sie in Abschnitt 6.

Technische Voraussetzungen sind opsi 4.0.1 mit den Paketständen:

Tabelle 5: Benötigte Pakete

opsi-Paket	**Version**
opsi-client-agent	>=4.0.1-23
opsi-winst	>=4.11.2.1
python-opsi	>=4.0.1.31-1

19.2 Einführung

Der *opsi-winst* verfügt über eine Reihe von speziellen Befehlen um Modifikationen in Profilen vorzunehmen. Diese Arbeiten aber auf den lokalen Profilen und sind beim Einsatz von *Roaming Profiles* (*Servergespeicherte Profile*) weitgehend nutzlos. Mit der opsi Erweiterung *User Profile Management* wird nun eine Möglichkeit geschaffen auch hier Veränderungen an den Profilen vorzunehmen. Dies geschieht in dem beim User Login der *opsi-winst* gestartet wird um spezielle *userLoginScripte* auszuführen.

19.3 Konzept

Wenn die Profile nicht bei der Installation der Software gleich mit gepatcht werden können, muss zwischen dem *Maschinen Teil* und dem *Profil Teil* der Installation deutlicher unterschieden werden. Die kann sowohl innerhalb eines Scriptes geschehen als auch durch die Auslagerung des *Profil Teils* in ein eigenes Script. Vielerorts passiert dies auch jetzt schon, in dem die *Profil Teile* im Rahmen eines Domain Login Scripts ausgeführt werden.

Je nach Praxis liegen daher die *Profil Teile* von opsi-Produkten als Bestandteil der opsi-scripte zur Installation und Deinstalltion vor, als auch als Bestandteil eines Domain Loginscriptes. Ziel dieser Erweiterung ist es, beide Varianten möglichst einfach in den neuen Mechanismus integrieren zu können.

Die Kernkonzepte dieser opsi Erweiterung sind:

- Ausführen spezieller userLoginScripte beim Login des users
  Im Rahmen des User Logins wird der *opsi-winst* gestartet aber in einem speziellem Modus ausgeführt in dem nur bei den Produkten hinterlegte *userLoginScripte* ausgeführt werden.

- Ausführen der Scripte mit administrativen Rechten aber im Userkontext
  Domain Login Scripte werden vom User mit user Rechten ausgeführt. Die opsi *userLoginScripte* werden vom *opsi-winst* ausgeführt, welcher mit administrativen Rechten läuft. Gleichzeitig begibt sich der *opsi-winst* aber in den Kontext des Users der sich eingelogged hat, so dass die Manipulation der Profile mit den selben Befehlen durchgeführt werden kann, wie in einem Domain Loginscript.
- Ausführen der Scripte innerhalb des opsi-service Kontext
  Die opsi *userLoginScripts* laufen innerhalb des opsi-service Kontextes und haben so über Scriptkonstanten die Informationen zu Produktnamen, Version und Packageversion die gerade bearbeitet wird. Weiterhin sind die Werte der Produktproperties im Zugriff sowie alle sonstigen Informationen welche eventuell über opsiservicalls abgerufen werden sollen.

Einschränkungen:

- Die *userLoginScripte* werden auch bei der Verwendung der opsi-WAN-Erweiterung nicht aus dem lokalen Cache geladen, sondern online.

19.4 Neue und erweiterte *opsi-winst* Funktionen

- Aufrufparameter `/allloginscripts`
  Wird der *opsi-winst* im opsi-service Kontext mit dem zusätzlichen Parameter */allloginscripts* aufgerufen, so hat das im wesentlichen folgende Auswirkungen:
  - Es werden die Produkte ermittelt welche ein *userLoginScript* haben. Nur für diese Produkte werden die *userLoginScripte* ausgeführt.
  - Es wird der user der sich eingelogt hat ermittelt und dafür gesorgt, dass die Konstanten zum aktuellen User wie z.B. `%CurrentAppdataDir%` auf die entsprechenden Verzeichnisse des eingelogten users zeigen. Ebenso werden Registry Operationen (`Registry` Sektionen und `GetRegistryString`) welche sich auf `HKCU` beziehen, so ausgeführt, das die Daten aus dem Registryzweig des Users kommen.
- Aufrufparameter `/silent`
  Der Aufrufparameter `/silent` sorgt dafür, das während der Scriptabarbeitung das Fenster des *opsi-winst* nicht angezeigt wird.
- Funktion `GetScriptMode`
  Um innerhalb eines Scriptes zu unterscheiden in welchem Modus das Script gerade ausgeführt wird, liefert die Funktion `GetScriptMode` zwei mögliche Werte zurück:
  - *Machine*
    Das Script wird **nicht** als *userLoginScript* ausgeführt (sondern z.B. als setup oder uninstall Script).
  - *Login*
    Das Script wird als *userLoginScript* ausgeführt.
- Neue primäre Sektion `ProfileActions`
  diese neue Sektion kann dazu dienen um Aktionen auf Userprofilen zusammenzufassen. Dabei kann ein Syntax verwendet werden, der es ermöglicht, diese Sektion sowohl als Bestandteil eines normalen Loginscripts als auch als *userLoginScript* zu nutzen. Dazu wird diese primäre Sektion auf unterschiedliche Art ausgewertet, je nachdem ob das script im Machine mode oder Login mode (also als userLoginScript) läuft.
  - *Login*
    Läuft ein Script als *userLoginScript* und enthält eine Sektion `ProfileActions`, so wird die Scriptabarbeitung bei dieser Sektion gestartet (und nicht bei `Actions`).
  - *Machine*
    Läuft ein Script als normales Installationsscript, so kann die Sektion `ProfileActions` ähnlich einer *Sub*-Sektion als Untersektion aufgerufen werden. Für die Abarbeitung dieser Sektion gilt: Für alle *Registry*-Sektions Aufrufe ist implizit */AllNtUserDats* gesetzt. Für alle *Files*-Sektions Aufrufe ist implizit */AllNtUserProfiles* gesetzt.
    Seit Version 4.11.3.2 gilt: Für alle *Patches*-Sektions Aufrufe ist implizit */AllNtUserProfiles* gesetzt.

- Registry Sektionen

  - Registry Sektionen welche auf *HKCU* bzw. *HKEY_CURRENT_USER* arbeiten, werden im Loginscript Mode so ausgeführt, dass die Änderungen im Zweig des eingeloggten users landen. Entsprechendes gilt für die Funktionen `GetRegistryStringValue*`.
  - Registry Sektionen welche im Normalen Modus (*Machine*) mit dem Modifier */AllNtUserDats* aufgerufen werden, dürfen jetzt in der `openkey` Anweisung den Root *HKCU* bzw. *HKEY_CURRENT_USER* enthalten. Dies ermöglicht es, die selbe Registry Sektion in den unterschiedlichen Modi auszuführen.

- Vermeidung unnötiger Läufe:
  Mit den Befehl `saveVersionToProfile` kann im aktuelle Profil hinterlegt werden, das das userLoginscript zu diesem Produkt in dieser Version gelaufen ist. Mit der Stringfunktion `readVersionFromProfile` bzw. der boolschen Funktion `scriptWasExecutedBefore` kann überprüft werden ob das userLoginScript zu diesem Produkt in dieser Version schon mal zuvor gelaufen ist und eine erneute Ausführung unnötig ist. Dazu liest diese Funktion zunächst einen evtl. vorhandenen Versionsstempel vom Profil ein (wie das mit `readVersionFromProfile` möglich ist) und vergleicht diesen mit der aktuell laufenden Version. Aus dem Vergleich ergibt sich der Rückgabewert (wahr/falsch). Danach werden noch die aktuellen Werte in das Profil zurückgeschrieben (wie das mit `saveVersionToProfile` möglich ist). Somit benötigen Sie nur diese eine Funktion in einer `if` Anweisung, um zu prüfen ob das Script schon mal gelaufen ist.
  Weiterhin liefert die Stringlistenfunktion `getProductMap` eine Infomap, aus der entnommen werden kann ob das aktuelle Produkt installiert oder deinstalliert usw. ist.

- Logging
  Die Logs von userLoginScripten werden geschrieben nach:
  `c:\opsi.org\log\<login user name>_login.log`
  Diese Logdateien werden auch an den opsi-server übertragen. Dabei wird eine neue Logdatei an eine existierende angehängt. Der opsi-server sorgt dafür, dass diese Dateien in der Größe beschränkt bleiben (max. 5 MB). Auf dem opsi server liegen diese logs unter `/var/log/opsi/userlogin/<clientid>.log`
  Im opsi Managementinterface (opsi-configed) werden diese Logs in einem zusätzliche Untertab *userlogin* in dem Tab *Logdateien* angezeigt.

19.5 Beispiele von userLoginScripten

Zunächst zwei Beispiele die so aufgebaut sind, wie sie auch in Domain Loginscripten eingesetzt werden könnten.

Ein sehr einfaches allgemeines Beispiel:

```
[Actions]
requiredWinstVersion >= "4.11.3.2"
Message "Example Profile Patch ...."

Files_profile_copy
Registry_currentuser_set
Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini"

[Files_profile_copy]
copy "%Scriptpath%\profiles\*.*" "%CurrentAppdataDir%\ACME"

[Registry_currentuser_set]
openkey [HKCU\Software\ACME]
set "show_greeting_window" = "no"

[Patches_profile_ini]
add [secdummy] dummy1=add1
```

Ein Beispiel zur Firefoxkonfiguration:

```
[Actions]
requiredWinstVersion >= "4.11.3.2"
Message "Firefox Profile Patch ...."

DefVar $akt_profile_ini$
DefVar $rel_prefs_path$

comment "check for existing profile ..."
Set $akt_profile_ini$ = "%CurrentAppdataDir%\Mozilla\Firefox\profiles.ini"
if FileExists($akt_profile_ini$)
        Set $rel_prefs_path$ = GetValueFromInifile($akt_profile_ini$,"Profile0","Path","")
        if FileExists("%CurrentAppdataDir%\Mozilla\Firefox\\"+$rel_prefs_path$)
                comment "We found the profile and will now patch it ....."
        endif
else
        comment "no firefox profile found for user"
endif
```

Als nächstes zeigen wir ein Beispiel welches das erste erweitert um die Möglichkeit Dinge aus dem Profil auch wieder zu entfernen. Je nachdem ob das Produkt auf dem Rechner installiert oder deinstalliert wird ein anderer Scriptteil ausgeführt:

```
[Actions]
requiredWinstVersion >= "4.11.3.2"
Message "Example Profile Patch ...."

if getValue("installationstate", getProductMap) = "installed"
        comment "Product is installed"
        Files_profile_copy
        Registry_currentuser_set
        Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini"
endif

if getValue("lastactionrequest", getProductMap) = "uninstall"
        comment "Product was uninstalled"
        Files_profile_del
        Registry_currentuser_del
endif

[Files_profile_copy]
copy "%Scriptpath%\profiles\*.*" "%CurrentAppdataDir%\ACME"

[Registry_currentuser_set]
openkey [HKCU\Software\ACME]
set "show_greeting_window" = "no"

[Files_profile_del]
del -s -f "%CurrentAppdataDir%\ACME"
del "%userprofiledir%\opsi-winst-test.ini"

[Patches_profile_ini]
add [secdummy] dummy1=add1

[Registry_currentuser_del]
deletekey [HKCU\Software\ACME]
```

Nun ein Beispiel welches das Setup Skript (setup32.ins und delsub32.ins) nutzt um unnötige Verdopplung des Codes zu vermeiden:

setup32.ins:

```
[Actions]
requiredWinstVersion >= "4.11.3.2"

DefVar $MsiId$
DefVar $UninstallProgram$
DefVar $ProductId$
DefVar $InstallDir$

; ----------------------------------------------------------------
; - Please edit the following values                             -
; ----------------------------------------------------------------
Set $ProductId$       = "ACME"
Set $InstallDir$      = "%ProgramFiles32Dir%\ACME"
; ----------------------------------------------------------------
        comment "Show product picture"
        ShowBitmap "%ScriptPath%\\" + $ProductId$ + ".png" $ProductId$

        if FileExists("%ScriptPath%\delsub32.ins")
                comment "Start uninstall sub section"
                Sub "%ScriptPath%\delsub32.ins"
        endif

if GetScriptMode = "Machine"
        Message "Installing " + $ProductId$ + " ..."

        comment "Start setup program"
        Winbatch_install

        comment "Patch the local Profiles ..."
        Registry_currentuser_set /AllNtUserDats
        Files_profile_copy /AllNtUserProfiles
        Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini" /AllNtUserProfiles
endif

if GetScriptMode = "Login"
        comment "login part"
        Files_profile_copy
        Registry_currentuser_set
        Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini"
endif


[Winbatch_install]
"%ScriptPath%\setup.exe" /sp- /silent /norestart

[Files_profile_copy]
copy "%Scriptpath%\profiles\*.*" "%CurrentProfileDir%\Appdata\ACME"

[Registry_currentuser_set]
openkey [HKCU\Software\ACME]
set "show_greeting_window" = "no"

[Patches_profile_ini]
```

```
add [secdummy] dummy1=add1
```

delsub32.ins:

```
Message "Uninstalling " + $ProductId$ + " ..."

if GetScriptMode = "Machine"
        comment "The machine part ..."
        Set $UninstallProgram$ = $InstallDir$ + "\uninstall.exe"
        if FileExists($UninstallProgram$)
                comment "Uninstall program found, starting uninstall"
                Winbatch_uninstall
        endif
        ; does also work since 4.11.2.1
        Registry_currentuser_del /AllNtUserDats
        Files_profile_del /AllNtUserProfiles
endif

if GetScriptMode = "Login"
        comment "The profile part ..."
        Files_profile_del
        Registry_currentuser_del
endif

[Winbatch_uninstall]
"$UninstallProgram$" /silent /norestart

[Files_profile_del]
del -s -f "%CurrentAppdataDir%\ACME"
del "%userprofiledir%\opsi-winst-test.ini"

[Registry_currentuser_del]
deletekey [HKCU\Software\ACME]
```

Nun ein Beispiel welches eine Variante des vorherigen Beispiels ist. Dabei wird der code durch die Verwendung der neuen primären Sektion `ProfileActions` vereinfacht und das Script ist sowohl als Installationsscript als auch als *userLoginScript* verwendbar.

```
[Actions]
requiredWinstVersion >= "4.11.3.2"

DefVar $ProductId$
DefVar $InstallDir$

Set $ProductId$       = "ACME"
Set $InstallDir$      = "%ProgramFiles32Dir%\ACME"

comment "Show product picture"
ShowBitmap "%ScriptPath%\\" + $ProductId$ + ".png" $ProductId$

Message "Installing " + $ProductId$ + " ..."

comment "Start setup program"
Winbatch_install

comment "Patch the local Profiles ..."
ProfileActions
```

```
[ProfileActions]
comment "login part"
Files_profile_copy
Registry_currentuser_set
Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini"

[Winbatch_install]
"%ScriptPath%\setup.exe" /sp- /silent /norestart

[Files_profile_copy]
copy "%Scriptpath%\profiles\*.*" "%CurrentProfileDir%\Appdata\ACME"

[Registry_currentuser_set]
openkey [HKCU\Software\ACME]
set "show_greeting_window" = "no"

[Patches_profile_ini]
add [secdummy] dummy1=add1
```

Nun eine Variante, welche sich im Profil merkt ob das Skript für dieses Produkt in dieser Version und diesen User schon mal ausgeführt wurde.

```
[Actions]
requiredWinstVersion >= "4.11.3.2"
Message "Example Profile Patch ...."

comment "Did we run this script before ? - and set version stamp in profile"
if not (scriptWasExecutedBefore)
        comment "loginscript was not run yet "
        Files_profile_copy
        Registry_currentuser_set
        Patches_profile_ini "%userprofiledir%\opsi-winst-test.ini"
endif

[Files_profile_copy]
copy "%Scriptpath%\profiles\*.*" "%CurrentAppdataDir%\ACME"

[Registry_currentuser_set]
openkey [HKCU\Software\ACME]
set "show_greeting_window" = "no"

[Patches_profile_ini]
add [secdummy] dummy1=add1
```

19.6 Konfiguration

Um die *User Profile Management* Erweiterung zu nutzen muss in der Konfiguration des opsiclientd das Loginevent aktiviert werden. Für dieses Event wird (wenn der entsprechend aktuelle opsi-client-agent auf dem Client installiert ist) der *opsi-winst* mit dem ergänzenden Parameter */alllogінscripts* gestartet werden.

Die Aktivierung des Loginevents können Sie auf der Kommandozeile wie folgt einrichten:

```
opsi-admin -d method config_createBool opsiclientd.event_user_login.active "user_login active" true
```

Als weiterer *opsi-winst* Parameter kann zusätzlich auch noch der Parameter */silent* verwendet werden, welcher die Anzeige des *opsi-winst* Fensters unterbindet.

```
opsi-admin -d method config_createUnicode opsiclientd.event_user_login.action_processor_command "user_login \
    action_processor" "%action_processor.command% /sessionid %service_session% /allloginscripts /silent" "%\
    action_processor.command% /sessionid %service_session% /alloginscripts /silent"
```

Die so eingerichteten Einstellungen können Sie im opsi Managementinterface im Tab *Hostparameter* Server- oder Client-spezifisch modifizieren.

19.7 Notification

Wenn Sie (wie oben beschrieben) das Loginevent aktiviert haben, so sehen Sie nach jedem Login den user_login_nofier:

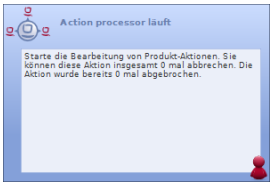


Abbildung 82: User Login Notifier

20 opsi-Nagios-Connector

20.1 Einführung

Neben dem Client Management ist das Monitoring der IT-Infrastruktur eine der Kernfunktion in der heutigen IT. opsi ist ein Client-Management Werkzeug. Für das Monitoring gibt es erprobte und bekannte Opensource Lösungen. Daher der Ansatz opsi nicht um ein Monitoring zu erweitern, sondern eine Schnittstelle zu bestehenden Lösungen anzubieten. Mit dieser Erweiterung von opsi wird eine Schnittstelle für die bekannte Monitoring Lösung Nagios zur Verfügung gestellt. In den folgenden Kapiteln wird der opsi-Nagios-Connector erläutert und Beispielkonfigurationen vorgestellt.

Der opsi-Nagios-Connector ist nicht fest an eine Monitoringsoftware gebunden. Programmiert wurde die Schnittstelle für Nagios/Icinga. Der Einsatz mit anderen Monitoringlösungen ist denkbar, wird momentan aber weder getestet noch unterstützt.

Die folgende Dokumentation beschreibt wie der opsi-Nagios-Connector aufgebaut ist und wie man ihn konfigurieren muss. Es wird vorausgesetzt, dass eine funktionierende Installation von Nagios bzw. Icinga vorliegt.

20.2 Vorbedingungen

20.2.1 Vorbedingungen bei opsi-Server und -Client

Dieses Modul ist momentan eine kofinanzierte opsi Erweiterung.
Es sind eine Reihe von Vorbedingungen nötig, um dieses Modul einsetzen zu können. Das bedeutet, dass Sie zum Einsatz eine Freischaltdatei benötigen. Diese Freischaltung erhalten Sie beim Kauf der Erweiterung. Zu Evaluierungszwecken stellen wir Ihnen auch eine zeitlich befristete Freischaltung kostenlos zur Verfügung (→ mail an info@uib.de).
Weitere Details hierzu finden Sie in Abschnitt 6.

Technische Voraussetzungen sind opsi 4.0.2 mit den Paketständen:

Tabelle 6: Benötigte Pakete

opsi-Paket	Version
opsi-client-agent	>=4.0.2-1
opsiconfd	>=4.0.2.1-1
python-opsi	>=4.0.2.1-1

20.2.2 Vorbedingungen beim Nagios Server

Vorrausgesetzt wird eine Nagios Installation in einer aktuellen 3.x Version oder eine Icinga Installation mindestens in der Version 1.6. Der opsi-Nagios-Connector sollte auch mit allen Nagios-Kompatiblen Monitoring-Lösungen benutzbar sein, getestet wird diese opsi Erweiterung allerdings nur gegen Nagios und Icinga. Sollte eine grafische Darstellung der opsiconfd-Performance Werte gewünscht werden, wird zusätzlich noch eine Installation des Addons pnp4nagios benötigt.

Für weitere Informationen siehe auch:

www.nagios.org

www.icinga.org

www.pnp4nagios.org

20.3 Konzept

Der opsi-Nagios-Connector besteht hauptsächlich aus zwei verschiedenen Komponenten. Im Folgenden werden beide Komponenten erläutert.

20.3.1 opsi-Webservice Erweiterung

Eines der Hauptschwerpunkte des opsi-Nagios-Connectors ist eine Erweiterung des opsi-Webservice. Die Webserviceschnittstelle bietet damit die Möglichkeit Checks vom opsi-Server anzufordern. Das bedeutet der Nagios Server verwendet die erweiterten Webservicemethoden zum Auslösen und Abfragen von Checks. Der Vorteil dieser Lösung ist, dass die eigentliche Checkausführung im opsi-System ausgeführt wird und somit der Aufwand auf den überwachten Systemen gering bleibt.

Diese Erweiterung stellt eine Sammlung von Checks zur Verfügung, die in einem späteren Kapitel einzeln vorgestellt werden. Hauptsächlich wurden opsi spezifische Checks eingebaut. "Normale" Systemchecks werden nicht über den opsi-Nagios-Connector angeboten und müssen auf konventionelle Weise ausgeführt werden.

20.3.2 opsi-client-agent Erweiterung

Ein weiterer Teil des opsi-Nagios-Connectors ist eine Erweiterung des opsi-client-agent. Da in einer opsi-Umgebung jeder Managed-Client einen laufenden opsi-client-agent hat, bietet es sich an, diesen als Nagios-Agent zu benutzen. Wichtig bei dieser Erweiterung ist zu wissen, dass nicht alle Funktionen eines Nagios-Agenten, wie z.B. `NSClient++` implementiert wurden.

Die Möglichkeiten des opsi-client-agent umfassen das Ausführen von Pluginbefehlen auf der Kommandozeile und das Zurückliefern der Ergebnisse.

Für Situationen in denen erweiterte Funktionen vom Nagios-Agent wie NSCA benötigt werden, wird ein opsi-Paket für die Verteilung und Pflege des `NSClient++` per opsi bereitgestellt.

Der Vorteil der Verwendung des opsi-client-agent ist zum Einen, dass kein zusätzlicher Agent benötigt wird und das der Monitoring-Server keine Zugangsdaten vom Client haben muss, da die Checks zum Client alle über den opsiconfd laufen. Dies vereinfacht auch die Konfiguration auf der Monitoring Seite um einiges.

20.4 opsi-Checks

Im folgenden Kapitel werden Zweck und Konfiguration der einzelnen opsi-Checks erklärt.

20.4.1 Hintergrund zum richtigen Verteilen der Checks

In der allgemeinen Monitoring Sprache wird zwischen aktiven und passiven Checks unterschieden. Die Bedeutung besagt, ob Nagios/Icinga die Checks selber auslöst und ausführt (aktiv) oder ob die Hosts die Checks eigenständig ausführen und die Ergebnisse an Nagios/Icinga selbstständig zurückmelden (passiv).

Auch bei der opsi-Nagios-Connector Erweiterung gibt es diese zwei Unterscheidungen. Allerdings werden diese bei opsi als direkte- bzw. indirekte-Checks bezeichnet:

- Direkt: Diese Checks werden auf dem Client ausgeführt und liefern Ergebnisse vom Client an den Monitoring Server.
- Indirekt: Die Erhebung der Daten, die für diese Art von Checks relevant sind, findet auf Basis von Backend-Daten im opsi-System statt. Diese können von realen Situationen und Zuständen abweichen.

Ein gutes Beispiel für einen indirekten Check ist `check-opsi-client-status`. Dieser Check bezieht sich eigentlich auf einen Client, ausgeführt wird er aber auf dem opsi-Backend. Der Client-Status ist in diesem Fall der Zustand des Clients aus Sicht von opsi. Im Gegensatz dazu wird jeder Check der tatsächlich am Client ausgeführt wird als direkter Check bezeichnet.

Die richtige Verteilung der Checks und damit die richtige Konfiguration erfordert zunächst die Analyse der eigenen opsi-Umgebung. Da opsi sehr flexibel einsetzbar ist, können verschiedene Szenarien auftreten. Hier werden die am häufigsten mit opsi eingesetzten Installationsvarianten abgehandelt. (Für spezielle Installation sollten Sie uns kontaktieren oder direkt über einen bestehenden Supportvertrag Ihre Situation schildern.)

Ein opsi-Server (Standalone Installation): Diese Variante ist die am häufigsten eingesetzten Variante. Bei dieser Installation ist der Configserver auch gleichzeitig der Depotserver und somit werden alle Checks über den opsi-Configserver aufgerufen.

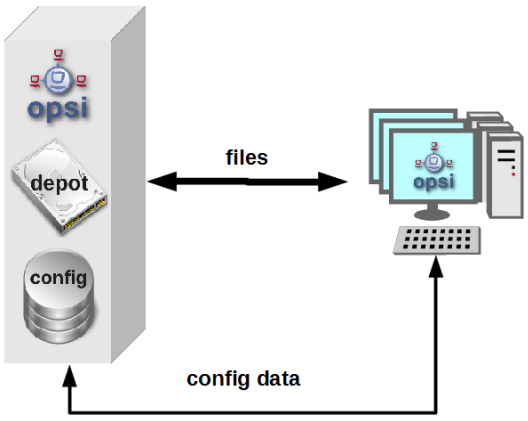


Abbildung 83: Schema eines standalone opsi-servers

Mehrere opsi-Server mit einer zentralen Administration (Multi-Depot Umgebung): Um bei dieser Art der Installation die Checks richtig zu verteilen muss man als erstes Verstehen, wie eine Multi-Depot Umgebung aufgebaut ist:

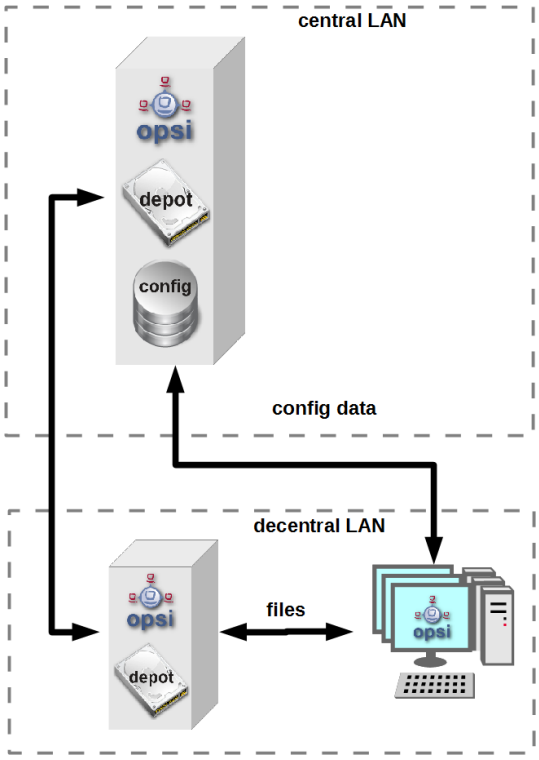


Abbildung 84: Schema einer opsi multidepot installation

Wie im Bild zu erkennen, gibt es nur ein Daten-Backend, welches am opsi-Configserver angesiedelt ist. Jeder Check, der gegen das Backend ausgeführt wird, muss somit zwangsläufig über den opsi-Configserver. Somit werden alle Checks die an die Depotserver gehen, intern an den Configserver weitergeleitet. Deshalb ist es sinnvoller diese Checks direkt gegen den opsi-Configserver aus zu führen. Eine Ausnahme können die aktiven Checks gegen den opsi-client-agent bilden. Wenn zum Beispiel zwischen den Servern eine Firewall aufgestellt ist, die nur den Port 4447 durchlässt, können Clients an der Außenstelle eventuell nicht erreicht werden (Standardport 4441). In solchen Fällen kann es nützlich sein den aktiven Check am Depotserver in der Außenstelle auszuführen.

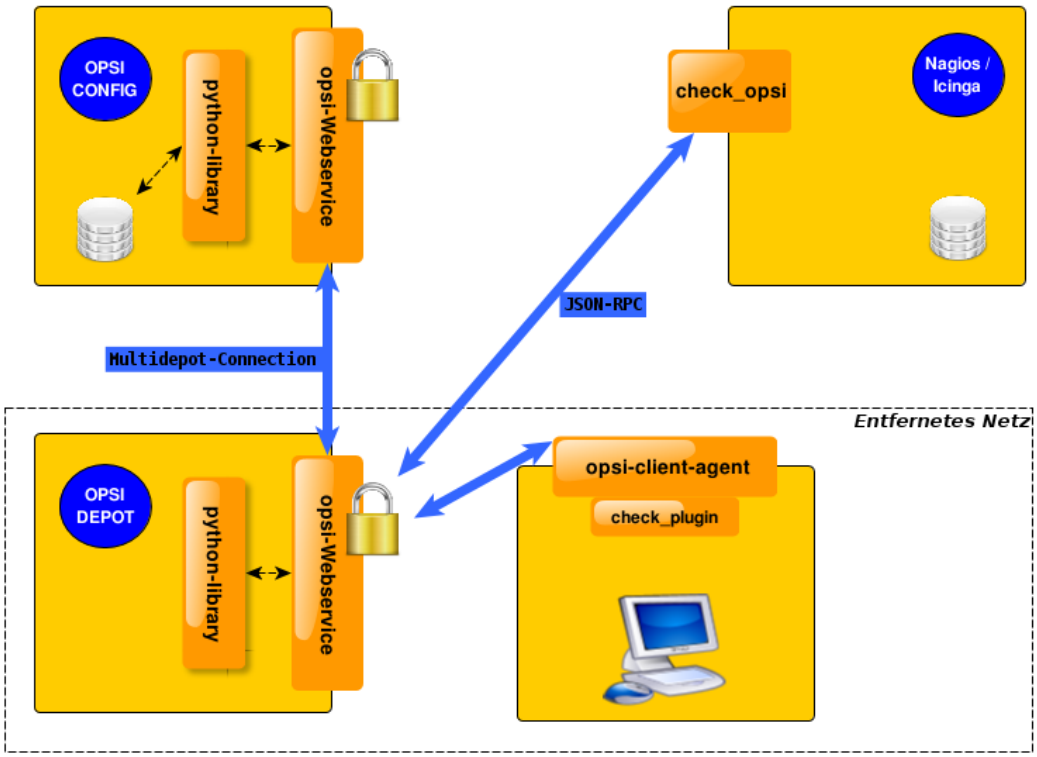


Abbildung 85: Verteilte Checks

20.4.2 opsi-check-plugin

Auf dem Nagios Server gibt es nur ein opsi-check-plugin, welches aber eine große Zahl von Checks unterstützt. Deshalb hat dieses Plugin auch ziemlich viele Optionen. Eine Auflistung dieser Optionen wäre der Erläuterung nicht dienlich, deshalb wird an dieser Stelle darauf verzichtet. Die einzelnen Optionen, die benötigt werden oder möglich sind, werden bei den einzelnen Checks erläutert. Das opsi-check-plugin wird aufgerufen mit dem Befehl `check_opsi`. Eine Übersicht der möglichen Optionen erhält man mit dem Parameter -h oder --help.
Die folgenden Optionen sind für alle Checks notwendig:

Tabelle 7: Allgemeine Optionen

Optionen	Bezeichnung	Beispiel
-H,--host	opsiServer auf dem gecheckt werden soll	configserver.domain.local
-P,--port	opsi-Webservice Port	4447 (Default)
-u,--username	opsi Monitoring User	monitoring
-p,--password	opsi Monitoring Password	monitoring123
-t,--task	opsi Checkmethode (Case Sensitive)	

Die oben aufgeführten Parameter müssen immer gesetzt werden. Das nachfolgende Kapitel erläutert als erstes, wie man das opsi-check-plugin manuell aufrufen würde. Wie diese über Nagios/Icinga gesetzt werden, wird im Kapitel der Konfiguration erläutert.

Um das check-plugin vom opsi-Nagios-Connector zu installieren, können Sie einfach das opsi-Repository auf dem Nagios-Server eintragen und mit folgendem Befehl das Paket installieren:

```
apt-get install opsi-nagios-plugins
```

Für Redhat/Centos:

```
yum install opsi-nagios-plugins
```

Für OpenSuse/SLES:

```
zypper install opsi-nagios-plugins
```

Das Plugin selbst ist in Python geschrieben und sollte auch auf anderen Distributionen laufen. Dieses Paket basiert auf dem Paket: *nagios-plugins-basic* bzw. *nagios-plugins* und installiert entsprechend das Plugin ins Verzeichnis: `/usr/lib/nagios/plugins`. Die Konfiguration der Nagios-Check-Commands wird nicht automatisch angelegt, da dieses Plugin sehr flexibel einsetzbar ist. Deshalb wird dieser Teil im Kapitel über die Konfiguration etwas später näher erläutert.

Check: opsi-Webservice

Mit diesem Check wird der opsi-Webservice-Prozess überwacht. Dieser Check liefert auch Performancewerte. Deshalb sollte dieser Check auf jedem opsi-Server selbst ausgeführt werden, da jeder opsi-Server seinen eigenen opsiconfd-Prozess hat, der überwacht werden kann bzw. sollte.

```
check_opsi.py -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiWebservice
```

Dieser Check liefert in der Regel OK zurück. In folgenden Situationen kann dies Abweichen:

- Critical: Wenn der Webservice ein Problem hat und nicht richtig antwortet. Weiterhin wird dieser Status zurückgemeldet, wenn die CPU-Last des Prozesses 80% überschreitet oder wenn die prozentuale Wert von RPC-Errors 20% erreicht und übersteigt (im Bezug auf die gesamten RPCs).
- Warning: Wenn der Webservice zwar arbeitet, aber Grenzwerte überschreitet: CPU Auslastung höher als 60% aber unter 80% oder wenn der prozentuale Wert der RPC-Errors in Bezug auf die Gesamt-RPC Anzahl höher als 10% aber unter 20% liegt.
- Unknown: Wenn der Webservice gar nicht erreichbar ist.

Info: Die CPU-Werte beziehen sich immer nur auf eine CPU, da ein Prozess auch nur einen Prozessor benutzen kann. Eine Ausnahme bildet hier das Multiprocessing von opsi.

Check: opsi-Webservice pnp4nagios-Template

Für die Perfomance-Auswertung gibt es ein Template für das pnp4nagios, welches die Werte kombiniert darstellt. Wie man das pnp4nagios installiert, wird hier nicht explizit beschrieben, sondern es wird davon ausgegangen, dass pnp4nagios richtig installiert und konfiguriert wurde. Die notwendige Vorgehensweise kann sich von der hier beschriebenen Lösung unterscheiden, wenn das pnp4nagios mit anderen Pfaden installiert wurde (kann bei selbst kompilierten Installationen vorkommen).

Es werden Standard-Templates verwendet, welche für jeden einzelnen Performancewert ein eigenes Diagramm erstellen. Um das oben genannte Template mit der kombinierten Ansicht zu verwenden, muss man folgendermaßen vorgehen:

Schritt 1: Erstellen Sie unterhalb von: `/etc/pnp4nagios/check_commands` eine Datei mit der Bezeichnung: `check_opsiwebservice.cfg` und folgendem Inhalt:

```
CUSTOM_TEMPLATE = 0
DATATYPE = ABSOLUTE,ABSOLUTE,ABSOLUTE,ABSOLUTE,DERIVE,GAUGE,GAUGE,GAUGE
```

Schritt 2: Legen Sie die Datei `check_opsiwebservice.php` unterhalb von `/usr/share/pnp4nagios/html/templates` ab. Diese Datei können Sie über svn.opsi.org auschecken.

```
cd /usr/share/pnp4nagios/html/templates
svn co https://svn.opsi.org/opsi-pnp4nagios-template/trunk/check_opsiwebservice.php
```

Wichtig bei diesen Templates ist, dass die Namen der PHP-Dateien genauso heißen wie der *command_name*, welcher in der Datei `/etc/nagios3/conf.d/opsi/opsicommands.cfg` definiert ist. Stimmen die Bezeichnungen nicht überein, wird ein Standardtemplate von pnp4nagios verwendet.

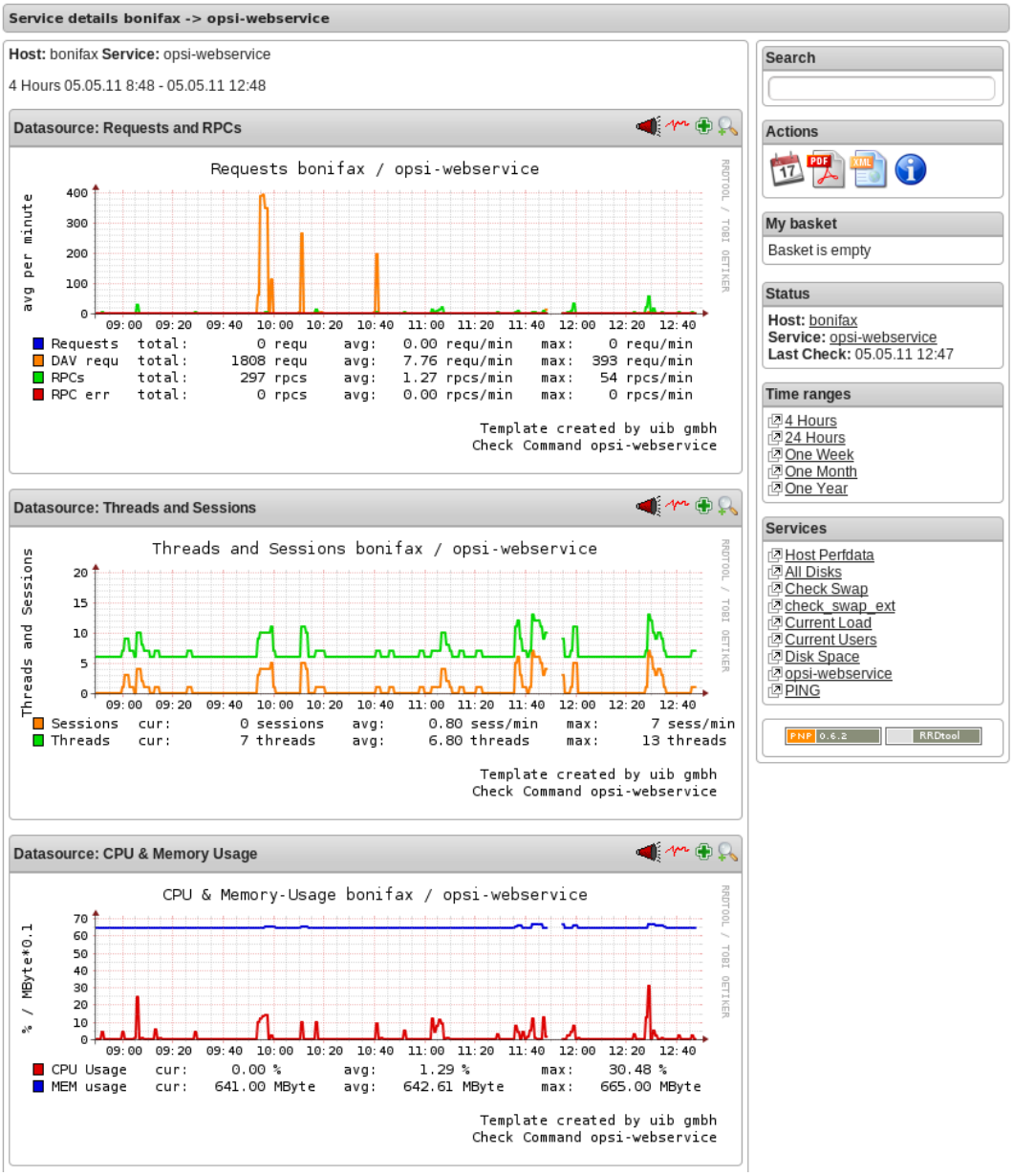
Sollte diese Anpassung vorgenommen werden, wenn die ersten Checks für den opsi-webservice schon stattgefunden haben, müssen die schon erstellten RRD-Datenbanken erst mal gelöscht werden, da mit diesen Templates auch die Struktur der RRD-Datenbanken neu konfiguriert werden.

In der Regel sind die RRD-Datenbanken unter folgendem Pfad zu finden: `/var/pnp4nagios/perfdata/<host>/` .

Dabei reicht es aus, alle Dateien, die entweder mit opsi-webservice.rrd oder mit opsi-webservice.xml beginnen, zu löschen. (Vorsicht: Hier werden auch andere RRD-Datenbanken von anderen Checks für diesen Host angelegt, die nicht unbedingt gelöscht werden sollten.).

Damit das Ganze automatisch funktioniert, müssen diese Dateien genauso heißen, wie das check_command vom opsi-Webservice. Der Grund dafür liegt in der Arbeitsweise von pnp4nagios. Sollte also die Konfiguration des opsi-Webservice von dieser Dokumentation abweichen, so müssen auch diese Template-Dateien umbenannt werden, da ansonsten pnp4nagios keine richtige Zuordnung treffen kann.

Wurde bis hierhin alles richtig konfiguriert und die Konfigurations-Schritte im Konfigurationskapitel richtig befolgt, sollten die Diagramme wie im folgenden Screenshot automatisch generiert werden:



Check: opsi-check-diskusage

Mit diesem Check werden die opsiconfd-Ressourcen auf Füllstand überwacht. Die folgende Tabelle zeigt, welche Ressourcen damit gemeint sind.

Tabelle 8: opsi Ressourcen

Ressource	Pfad
/	/usr/share/opsiconfd/static
configed	/usr/lib/configed
depot	/opt/pcbin/install
repository	/var/lib/opsi/repository

Bitte auch hier beachten, dass nur die Füllstände der Pfade, die für opsi relevant sind, beim Check berücksichtigt werden. Dieser Check soll keinen allgemeinen DiskUsage-Check ersetzen.

Mit folgendem Befehl kann man alle Ressourcen gleichzeitig abfragen:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDiskUsage
```

Zusätzlich zu der normalen Checkvariante gibt es die Möglichkeit nur eine Ressource zu checken. Das folgende Beispiel checkt nur nach der Ressource `depot`:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDiskUsage -r depot
```

Standardmäßig gibt dieser Check OK zurück und gibt den freien Speicherplatz der Ressource oder der Ressourcen zurück. Die Einheit ist dabei Gigabyte. Folgende weitere Zustände sind möglich:

- WARNING: Wenn eine oder mehrere Ressourcen 5GB oder weniger freien Speicherplatz haben.
- CRITICAL: Wenn eine oder mehrere Ressourcen 1GB oder weniger freien Speicherplatz haben.

Die oben genannten Werte sind die Standard-Thresholds. Diese können durch zusätzliche Angabe von den Optionen -C und -W bzw. --critical und --warning selber bestimmt werden. Dabei gibt es zwei mögliche Einheiten: 10G bedeutet mindestens 10 Gigabyte freier Speicherplatz, durch diese Angabe wird der Output auch in dieser Datei ausgegeben. Wenn die Thresholds in Prozent oder ohne Angaben angegeben werden, wird auch der Output in Prozent generiert. Abschließend noch ein Beispiel zur Veranschaulichung:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDiskUsage -r depot --warning\
      10% --critical 5%
```

Check: opsi-client-status

Einer von zwei Schwerpunkten des opsi-Nagios-Connectors ist das Überwachen von Software-Rollouts. Dafür wurde dieser Check konzipiert. Er bezieht sich immer auf einen Client innerhalb von opsi.

Der Status der Produkte auf dem Client entscheidet über das Ergebnis des Checks. Auf diese Weise erkennt man schnell, ob eine Produktinstallation auf dem betreffenden Client ansteht oder aktuell ein Problem verursacht hat. Aber nicht der Produktstatus des Clients kann das Checkergebnis beeinflussen, sondern auch wann dieser Client sich das letzte mal am Service gemeldet hat. Wenn der Client länger als 30 Tage nicht am Service war, wird der Check mindestens ein Warning als Ergebnis zurückgeben. Dieser Zeitraum orientiert sich an der Abfolge der Microsoft Patchdays. Wenn ein Client länger als 30 Tagen von der Softwareverteilung abgeschottet ist, sollte man den Client checken. Auf diese Weise kann man auch Clients im System finden, die schon lange inaktiv sind, die man im normalen Betrieb schnell übersieht.

Die Ergebnisse der Tests werden von folgenden zwei Grundregeln bestimmt:

- Der Software rollout Status ist:
  - *OK*
    wenn die Software auf dem Client in der selben Produkt- und Paketversion installiert ist, welche auf dem Server liegt und kein *Action Request* gesetzt ist.
  - *Warning*
    wenn die auf dem Client installierte Software von der Version auf dem Server abweicht oder ein *Action Request* gesetzt ist.
  - *Critical*
    wenn die letzte Aktion für die Software auf dem Client ein *failed* zurückgeliefert hat.
- Die Zeit seit *last seen* liefert:
  - *OK*
    wenn der Client vor 30 Tagen oder weniger gesehen wurde.

– *Warning*
wenn der Client vor mehr als 30 Tagen zum letzten Mal gesehen wurde.

Dieser Check kann auf verschiedene Weisen durchgeführt werden; die einfachste Variante:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkClientStatus -c opsiclient.\
    domain.local
```

Man kann einzelne Produkte anhand ihre ID von diesem Check ausschließen. Dies würde zum Beispiel so aussehen:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkClientStatus -c opsiclient.\
    domain.local -x firefox
```

Das obige Beispiel schließt das Produkt *firefox* aus diesem Check aus, somit würde dieser Check auch ein OK liefern, selbst wenn das Produkt *firefox* bei dem Client auf *failed* steht.

Check: opsi-check-ProductStatus

Der zweite Schwerpunkt des opsi-Nagios-Connectors ist das Überwachen von Software-Rollouts. Mit die wichtigste Aufgabe in einem Software-Verteilungssystem. Sobald eine neue Software, egal ob Standard-Software oder eigene Software mit opsi verwaltet, verteilt und aktuell gehalten wird, muss man den Rollout-Status im Auge behalten.

Das Ergebnis dieses Checks wird durch die folgenden Grundregeln bestimmt:

Der Software Rollout Status ist:

- *OK*
  wenn die Software auf dem Client in der selben Produkt- und Paketversion installiert ist, welche auf dem Server liegt und kein *Action Request* gesetzt ist.
- *Warning*
  wenn die auf dem Client installierte Software von der Version auf dem Server abweicht oder ein *Action Request* gesetzt ist.
- *Critical*
  wenn die letzte Aktion für die Software auf dem Client ein *failed* zurückgeliefert hat.

Durch einige Parameter kann man diesen Check flexibel einsetzen. Um dies besser zu verdeutlichen werden einige Beispielaufrufe aufgeführt und erläutert. Im einfachsten Fall ruft man den Check einfach für ein Produkt auf. Dieser wird als `productId` (opsi-interner Name) angegeben.

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -e firefox
```

In einer normalen opsi-Umgebung reicht dieser Aufruf, um den Zustand für das Produkt `firefox` zu überwachen. Die Ausgabe zeigt an, ob alles in Ordnung ist oder wie viele Installationen anstehen (setup), wie viele Clients ein Problem mit diesem Produkt haben (failed) und wie viele Clients nicht die aktuelle Version installiert haben.

Dies ist zur Übersicht in den meisten Fällen schon ausreichend, wenn man aber nun genau wissen will, auf welchen Clients, welcher Zustand zu diesem Produkt zu diesem Checkergebnis geführt hat, kann man den Befehl im `verbosemode` ausführen:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -e firefox -v
```

Bei einer Multi-Depot Umgebung wird der obige Befehl aber nicht die komplette Umgebung nach diesem Produkt überwachen, sondern nur den Configserver. (Oder exakter: Die Clients die dem depot auf dem config server zugewiesen sind). Wenn man mehrere Depotserver hat, kann man das Depot auch direkt mit angeben:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -e firefox -d \
    depotserver.domain.local
```

Der Grund dafür ist, dass jeder Depotserver diese Produkt zur Verteilung haben kann. Dies muss nicht überall die selbe Version sein, auch wenn die productId genau die Selbe ist. Aus diesem Grund müssen alle Clients anders beurteilt werden, je nachdem an welchem Depot sie registriert sind. Dies hat zusätzlich noch den Vorteil, dass man diesen Check später im Nagios beim Depotserver ansiedeln kann, was zusätzlich die Übersichtlichkeit erhöht. Wenn man nur ein oder zwei Depotserver hat, kann man auch mit der Angabe von `all` alle Depotserver mit einem Check abdecken oder Komma separiert mehrere Depotserver angeben.

Zusätzlich kann man mit diesem Check auch mit opsi-Gruppen arbeiten. Man kann zum Beispiel eine ganze Produktgruppe mit einem Check abfragen. Wenn man zum Beispiel eine Produktgruppe: `buchhaltung` bildet, kann man mit folgendem Aufruf:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -g buchhaltung
```

Nun werden alle Produkte, die Mitglieder in dieser Produktgruppe sind über diesen Check überwacht. Die Auswertung dieser Gruppe findet immer während des eigentlichen Checks statt; das bedeutet, man kann über opsi diese Gruppe bearbeiten und beeinflusst somit direkt die Check-Parameter ohne das man die Nagios-Konfiguration anpassen muss.

Anmerkung
Gruppen innerhalb von Gruppen werden nicht beachtet und müssen separat angegeben werden.

Auch die Clients, die für diesen Check abgearbeitet werden, können beeinflusst werden. Wie vorher schon erwähnt, wird die Clientliste durch Angabe des Depotservers beeinflusst, zusätzlich können opsi-Hostgruppen angegeben werden, die für diesen Check abgearbeitet werden. Dieser Aufruf sieht zum Beispiel wie folgt aus:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -g buchhaltung -G \
    produktivclients
```

Dies würde die Produkte der Produktgruppe *buchhaltung* für alle Clients der Gruppe `produktivclients` überprüfen. Auch bei den Hostgruppen gilt die Regel, dass Untergruppen dieser Gruppe nicht abgearbeitet werden. Für opsi-Productgroups gilt genauso, wie für opsi-Hostgroups, dass mehrere Gruppen Komma separiert angegeben werden können.

Abschließend kann man auch bei diesem Check opsi-Clients ausschließen:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkProductStatus -x client.domain.\
    local
```

Check: opsi-check-Depotsync

Gerade in einer Multi-Depot Umgebung ist es wichtig, die Depotserver auf Synchronität zu überwachen. Entscheidend ist bei diesem Check die Software- und Paketversion der installierten Produkte. Manchmal ist ein differenzierter Einsatz der opsi-Produkte auf Depotservern gewünscht, birgt aber die Gefahr, dass bei einem Umzug von einem Client, von einem Depot zum anderen, Inkonsistenzen in der Datenbank entstehen können. Um dieser Problematik entgegen zu wirken, wird empfohlen die opsi-Pakete auf den Depotservern so synchron wie möglich zu halten.

Standardmäßig liefert dieser Check *OK* zurück, sollte eine Differenz festgestellt werden, wird der Status: *WARNING* zurückgegeben. Dieser Check ist ein klassischer Check, der auf dem Configserver ausgeführt werden sollte, da alle Informationen zu diesem Check nur im Backend auf dem opsi-Configserver zu finden sind.

Als nächstes folgen ein paar Anwendungsmöglichkeiten dieses Checks:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus
```

Dies ist dies Basis-Variante und äquivalent zu folgendem Aufruf:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d all
```

Ohne konkrete Angabe von Depotserver werden die Produkt-Listen aller Depot-Server miteinander verglichen. Um eine bessere Übersichtlichkeit zu schaffen, sollte man diesen Check auf zwei Depotserver reduzieren und lieber auf mehrere Checks verteilen. Dies erreicht man durch direkte Angabe der Depotserver:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d \
    configserver.domain.local,depotserver.domain.local
```

Mit diesem Aufruf werden alle Produkte verglichen, die auf beiden Depotservern installiert sind. Sollte ein Produkt auf einem Depotserver gar nicht installiert sein, hat dies keine Auswirkungen auf das Check-Resultat. Dies kann man ändern, indem man bei diesem Check den "strictmode" Schalter setzt:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d \
    configserver.domain.local,depotserver.domain.local --strictmode
```

Nun werden auch Produkte angezeigt, die auf einem Depotserver nicht installiert sind. Um ein bestimmtes Produkt oder bestimmte Produkte nicht mit zu checken, weil man zum Beispiel will dass diese Produkte in verschiedenen Versionen eingesetzt werden, kann man diese Produkte von diesem Check ausschließen:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d \
    configserver.domain.local,depotserver.domain.local --strictmode -x firefox,thunderbird
```

Dieser Check würde auch dann ein OK zurückgeben, wenn das *firefox* und das *thunderbird*-Paket nicht überall synchron eingesetzt werden.

Ein weitere Einsatzmöglichkeit wäre, dass nur eine Auswahl von Produkten auf Synchronität überwacht werden können. Dies kann man durch:

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkOpsiDepotSyncStatus -d \
    configserver.domain.local,depotserver.domain.local --strictmode -e firefox,thunderbird
```

So werden nur *firefox* und *thunderbird* Produkte auf Synchronität überwacht. Bei diesem Check sollte der `strictmode` gesetzt sein, damit man auch erkennt, wenn die gewünschten Produkte auf Depotservern nicht installiert sind.

Check: Plugin über OpsiClientd checken

Dieser Check führt ein Check-Plugin auf dem Client direkt aus und fängt die Ausgabe ein.

Diese Erweiterung soll keinen Ersatz für einen richtigen Nagios-Agent bieten, sondern eine Alternative. Man kann diese Erweiterung einsetzen, wenn man Plugins auf dem opsi-Client checken will. Die eingesetzten Plugins müssen den sogenannten: `Nagios plug-in development guidelines` entsprechen. (Weitere Infos unter: http://nagiosplug.sourceforge.net/developer-guidelines.html).

Um ein Plugin ausführen zu können, muss man das Plugin erst einmal auf den Clients verteilen. Dies sollte man über ein opsi-Paket lösen. Der Ablageort für die Plugins auf dem Client ist im ersten momentan egal, da man den Pfad beim Checken mit angeben muss. Allerdings sollte man die Plugins nicht einzeln verteilen, sondern in einem Verzeichnis zusammenführen, damit das Aktualisieren und Pflegen der Plugins einfacher wird. Weiterhin sollte man auch Sicherheitstechnisch im Hinterkopf behalten, dass die Plugins im Systemkontext des opsiclientd-Services aufgerufen werden. Normale Anwender sollten auf dieses Verzeichnis keinen Zugang haben.

Es gibt diverse Plugins, die es schon vorgefertigt im Internet runter zu laden gibt. Eine mögliche Anlaufstelle ist http://exchange.nagios.org/ .

Im Folgenden wird davon ausgegangen, dass unter `C:\opsi.org\nagiosplugins\` das Plugin `check_win_disk.exe` vom Paket *nagioscol* (http://sourceforge.net/projects/nagiosplugincol/) abgelegt wurde.

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkPluginOnClient --plugin "C:\\\
    opsi.org\\nagiosplugincol\\check_win_disk.exe C:" -c client.domain.local
```

Dieser Aufruf checkt auf dem Client `client.domain.local` das Plugin check_win_disk.exe und übergibt diesem den Parameter `C:`. Dies bedeutet, dass das Laufwerk *C* auf dem Client gecheckt wird. Die Ausgabe und der Rückgabewert dieses Plugins wird direkt vollständig ausgewertet und diese Werte können in Nagios unmittelbar weiter verarbeitet werden.

Ein besonderes Feature ist das Beibehalten von Zuständen. Diese Implementation ist aus der Problemstellung entstanden, dass Clients nicht wie Server durchlaufen, sondern in der Regel nur einen bestimmten Zeitraum eingeschaltet

sind. Man kann den Check auf Nagios-Seite zwar mit sogenannten `Timeperiods` eingrenzen, aber in der Praxis ist so ein Vorgehen nicht praktikabel, da man zum Beispiel auch bei Urlaub von Anwendern flexibel reagieren muss. Dies würde eine ständige Konfigurationsarbeit nach sich ziehen. Wenn man darauf verzichtet, wird der Status ständig geändert, auch wenn ein aufgetretenes Problem noch gar nicht gelöst ist. Deshalb kann man den letzten bekannten Status an opsi übergeben. Sollte der Client nicht erreichbar sein, wird dieser letzte bekannte Status zurückgegeben. Ein *Critical*-Zustand bleibt beispielsweise in diesem Falle auch auf *Critical* stehen und wechselt nicht auf *Unknown*, was rein logisch aber richtig wäre.

Um dieses Feature später mit Nagios zu verwenden, kann man die Nagios-Makros: *$SERVICESTATEID$* und *$SERVICEOUTPUT$* nutzen.

```
check_opsi -H configserver.domain.local -P 4447 -u monitoring -p monitoring123 -t checkPluginOnClient --plugin "C:\\\
    opsi.org\\nagiosplugincol\\check_win_disk.exe C:" -c client.domain.local -s $SERVICESTATEID$ -o  $SERVICEOUTPUT$
```

20.5 opsi Monitoring Konfiguration

Dieses Kapitel widmet sich der Konfiguration der Schnittstelle von opsi und dem Nagios-Server. Die Konfigurationen in diesem Kapitel, besonders die auf dem Nagios-Server, sollen als Empfehlungen gelten, sind aber nicht die einzigen Lösungen. Hier wird nur die Konfiguration mit einem Nagios-Server beschrieben. Mit einem Icinga-Server sollte, mit Ausnahme von ein paar Pfaden, die Konfiguration ziemlich genauso funktionieren. Andere Derivate auf Nagios Basis sollten funktionieren, wurden aber nicht getestet.

Tipp

Die Konfigurationsdateien aus diesem Kapitel sind Bestandteil des opsi-nagios-connector-utils svn-Repository. Um die Beispiel-Konfigurationsdateien direkt zu beziehen können Sie auf dieses Repository per Browser zugreifen:

```
https://svn.opsi.org/listing.php?repname=opsi-nagios-connector-utils
```

oder direkt per svn ein Checkout ausführen:

```
svn co https://svn.opsi.org/opsi-nagios-connector-utils
```

20.5.1 opsi Monitoring User

In der Regel wird im Monitoring-Bereich viel mit IP-Freischaltungen als Sicherheit gearbeitet. Da aber dieser Mechanismus nicht wirklich einen Schutz bietet, wurde beim opsi-Nagios-Connector darauf verzichtet. Aus diesem Grund wird das Ganze per Benutzer und Passwort geschützt. Diesen User als opsi-admin einzurichten, würde aber auch hier zu viele Rechte freischalten, da dieser User nur für diese Schnittstelle von Nöten ist und auch die Benutzbarkeit auf diesen Bereich eingeschränkt werden soll, wird der User nur intern in opsi eingerichtet. Folgender Befehl legt den User an:

```
opsi-admin -d method user_setCredentials monitoring monitoring123
```

Dieser Befehl legt den User: monitoring mit dem Passwort monitoring123 an. Der User wird in der `/etc/opsi/passwd` angelegt und ist auch kein User, mit dem man sich an der Shell anmelden könnte.

Bei einer Multi-Depot Umgebung muss man diesen User nur auf dem Configserver erzeugen.

Beim Nagios-Server kann man dieses Passwort vor den CGI-Skripten maskieren, indem man einen Eintrag in der `/etc/nagios3/resource.cfg` vornimmt. Dieser sieht zum Beispiel so aus:

```
$USER2$=monitoring
$USER3$=monitoring123
```

Die Zahl hinter *$USER* kann variieren. Wenn diese Datei vorher nicht genutzt wurde, sollte in der Regel nur das `$USER1$` belegt sein. Diese Konfiguration dient als Grundlage für die weiteren Konfigurationen.

20.5.2 opsi Nagios-Connector Konfigurationsverzeichnis

Aus Gründen der Übersichtlichkeit empfiehlt es sich für die Konfigurationsdateien des opsi-Nagios-Connectors ein eigenes Verzeichnis anzulegen. Erzeugen Sie dazu unterhalb von: `/etc/nagios3/conf.d` ein Verzeichnis `opsi`. Dies macht die Konfiguration später übersichtlicher, da alle Konfigurationen, die den opsi-Nagios-Connector betreffen, gebündelt auf dem Server abgelegt sind.

Zu den Konfigurationsdateien in diesem Verzeichnis gehören:

- Nagios Template: `opsitemplates.cfg`
- Hostgroups: `opsihostgroups.cfg`
- Server Hosts: `<full name of the server>.cfg`
- Kommandos: `opsicheckcommands.cfg`
- Kontakte: `opsicontacts.cfg`
- Services: `opsiservices.cfg`

Um die Übersichtlichkeit noch weiter zu erhöhen sollten Sie unterhalb von `/etc/nagios3/conf.d/opsi` noch ein Verzeichnis `clients` anlegen, dass im Weiteren dazu dient, die Konfigurationsdateien für die Clients aufzunehmen.

20.5.3 Nagios Template: `opsitemplates.cfg`

Es gibt diverse Templates für diverse Objekte im Nagios. Dies ist eine Standard-Nagios Funktionalität, die hier nicht näher beschrieben wird. Diese Funktionalität kann man sich für opsi zu nutze machen, um sich später bei der Konfiguration die Arbeit zu vereinfachen.

Da die meisten Checks auf dem Configserver ausgeführt werden, sollte man sich als erstes ein Template für die opsi-Server und ein Template für die opsi-Clients schreiben. Da es in einer Multi-Depot Umgebung nur einen Configserver geben kann, kann man direkt diesen ins Template übernehmen. Dies erreicht man am einfachsten über eine Custom-Variable. Diese erkennt man daran, dass sie mit einem `_` beginnen. Es wird in das Template für den opsi-Server und die opsi-Clients folgendes zusätzlich eingetragen:

```
_configserver           configserver.domain.local
_configserverurl        4447
```

Auf diese beiden *custom Variablen* kann man später einfach mit dem Nagios-Makro: `$_HOSTCONFIGSERVER$` und `$_HOSTCONFIGSERVERPORT$`, verweisen. HOST muss vorher angegeben werden, da diese Custom-Variablen in einer Hostdefinition vorgenommen wurden. Weitere Informationen zu Custom-Variablen entnehmen Sie bitte der Nagios-Dokumentation. Da diese beiden Konfigurationen im Template vorgenommen werden müssen, gelten sie für jede Hostdefinition, die später von diesen Templates erben. Aber dazu später mehr.

Um nun die Templates anzulegen, sollte man unterhalb von: `/etc/nagios3/conf.d` als erstes ein Verzeichnis `opsi` erstellen. Dies macht die Konfiguration später übersichtlicher, da alle Konfigurationen, die den opsi-Nagios-Connector betreffen, gebündelt auf dem Server abgelegt sind. In diesem Verzeichnis erstellt man die Datei und benennt sie direkt so, dass man später erkennt, was genau hier konfiguriert werden soll: `opsitemplates.cfg`.

In dieser Datei können verschiedene Templates definiert werden. Die Templatedefinition richtet sich dabei nach folgendem Muster, bei dem zur besseren Verständlichkeit die Kommentare zu den einzelnen Einstellungen nicht gelöscht wurden:

```
define host{
        name                    opsihost-tmp    ; The name of this host template
        notifications_enabled           1       ; Host notifications are enabled
        event_handler_enabled           1       ; Host event handler is enabled
        flap_detection_enabled          1       ; Flap detection is enabled
        failure_prediction_enabled      1       ; Failure prediction is enabled
        process_perf_data               0       ; Process performance data
        retain_status_information       1       ; Retain status information across program restarts
```

```
        retain_nonstatus_information    1       ; Retain non-status information across program restarts
                max_check_attempts              10
                notification_interval           0
                notification_period             24x7
                notification_options            d,u,r
                contact_groups                  admins
        register                        0       ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL HOST, JUST A TEMPLATE!
        icon_image                      opsi/opsi-client.png
        }
```

- Optional kann durch die Option icon_image ein Image gesetzt werden, dieser muss relativ zum Pfad: `/usr/share/nagios3/htdocs/images/logos/` angegeben werden.
- Optional kann auch eine eigene *contact_group* angegeben werden, die allerdings als Contact-Object z.B. in der `opsicontacts.cfg` angelegt sein muss.

Wir empfehlen Templates für folgende Objekte anzulegen:

- opsi server
- opsi client
- opsi service
- sowie 2 templates für pnp4nagios (host-pnp / srv-pnp)

Zunächst das Beispiel des opsi-Server-Templates:

```
define host{
        name                            opsi-server-tmpl
        notifications_enabled           1
        event_handler_enabled           1
        flap_detection_enabled          1
        failure_prediction_enabled      1
        process_perf_data               1
        retain_status_information       1
        retain_nonstatus_information    1
                check_command                   check-host-alive
                max_check_attempts              10
                notification_interval           0
                notification_period             24x7
                notification_options            d,u,r
                contact_groups                  admins,opsiadmins
        _configserver                   configserver.domain.local
        _configserverport               4447
        register                        0
        icon_image                      opsi/opsi-client.png
        }
```

Hier muss natürlich noch *configserver.domain.local* an die lokalen Gegebenheiten angepasst werden. Auch die *contact_groups* bedürfen evtl. der Anpassung.

Als nächster Teil der Datei `opsitemplates.cfg` das Template für die Clients:

```
define host{
        name                            opsi-client-tmpl
        notifications_enabled           1
        event_handler_enabled           1
        flap_detection_enabled          1
        failure_prediction_enabled      1
        process_perf_data               1
        retain_status_information       1
        retain_nonstatus_information    1
                max_check_attempts              10
```

```
                notification_interval           0
                notification_period             24x7
                notification_options            d,u,r
                contact_groups                  admins,opsiadmins
        _configserver                   configserver.domain.local
        _configserverport               4447
        register                        0
        icon_image                      opsi/opsi-client.png
        }
```

Da die Clients in der Regel nicht durchlaufen, sollte die Option: "check command check-host-alive" nicht gesetzt werden. Somit werden die Clients als Pending angezeigt und nicht als Offline.

Hier muss natürlich noch *configserver.domain.local* an die lokalen Gegebenheiten angepasst werden. Auch die *contact_groups* bedürfen evtl. der Anpassung.

Als nächster Teil der Datei `opsitemplates.cfg` das Template für die opsi-services:

```
define service{
        name                            opsi-service-tmpl
        active_checks_enabled           1
        passive_checks_enabled          1
        parallelize_check               1
        obsess_over_service             1
        check_freshness                 0
        notifications_enabled           1
        event_handler_enabled           1
        flap_detection_enabled          1
        failure_prediction_enabled      1
        process_perf_data               1
        retain_status_information       1
        retain_nonstatus_information    1
                notification_interval           0
                is_volatile                     0
                check_period                    24x7
                normal_check_interval           5
                retry_check_interval            1
                max_check_attempts              4
                notification_period             24x7
                notification_options            w,u,c,r
                contact_groups                  admins,opsiadmins
        register                        0
        }
```

Wenn pnp4nagios für die grafische Darstellung der Performancedaten für den opsi-Webservice eingesetzt werden, sollten noch die folgenden zwei Objekte als Template in der Datei `opsitemplates.cfg` angelegt werden:

```
define host {
   name       host-pnp
   action_url /pnp4nagios/index.php/graph?host=$HOSTNAME$&srv=_HOST_
   register   0
}

define service {
   name       srv-pnp
   action_url /pnp4nagios/index.php/graph?host=$HOSTNAME$&srv=$SERVICEDESC$
   register   0
}
```

20.5.4 opsi Hostgroup: `opsihostgroups.cfg`

Als nächstes sollten folgende Hostgruppen erstellt werden. Dies dient zum einen der besseren Übersicht auf der Weboberfläche und hilft beim Konfigurieren der Services. Dafür sollte eine Datei Namens `opsihostgroups.cfg` mit folgendem Inhalt erstellt werden:

```
define hostgroup {
        hostgroup_name  opsi-clients
        alias           OPSI-Clients
}

define hostgroup {
        hostgroup_name  opsi-server
        alias           OPSI-Server
        members         configserver.domain.local, depotserver.domain.local
}
```

20.5.5 opsi Server: `<full name of the server>.cfg`

Als nächstes sollten die opsi-Server konfiguriert werden. Dies kann jeweils in einer eigenen Datei wie zum Beispiel configserver.domain.local.cfg oder eine Datei mit allen opsi-Hosts wie opsihost.cfg erfolgen. Der Inhalt sollte für opsi-Server folgendermaßen aussehen:

```
define host{
        use                             opsi-server-tmpl
        host_name               configserver.domain.local
        hostgroups              opsi-server
        alias                           opsi Configserver
        address                 configserver.domain.local
        }

define host{
        use                             opsi-server-tmpl
        host_name               depotserver.domain.local
        hostgroups              opsi-server
        alias                           opsi Depotserver
        address                 depotserver.domain.local
        }
```

Folgende Erläuterungen zu den Werten: * *use* bezeichnet, welches Template benutzt wird. * *hostgroups* gibt an, welcher Hostgruppe der Server angehört.

20.5.6 opsi Clients: `clients/<full name of the client>.cfg`

Die opsi-Clients sollten mindestens folgendermaßen definiert werden:

```
define host{
        use                             opsi-client-tmpl
        host_name               client.domain.local
        hostgroups              opsi-clients
        alias                           opsi client
        address                 client.domain.local
        _depotid                        depotserver.domain.local
        }
```

Die Clientkonfiguration bedient sich einer Sonderfunktion von Nagios. Die Option `_depotid` ist eine sogenannte benutzerdefinierte Variable, die als Makro `$_HOSTDEPOTID$` benutzt werden kann. Die Verwendung ist optional und wird verwendet, wenn die Ausführung eines direkten Checks nicht vom config server, sondern vom hier definierten Depotserver aus startet.

Um die Erstellung von vielen Client Konfigurationsdateien zu vereinfachen, können diese auf dem opsi-Config-Server mit folgendem Skript erstellt werden.

```
#!/usr/bin/env python

from OPSI.Backend.BackendManager import *

template = '''
```

```
define host {
        use             opsi-client-tmpl
        host_name       %hostId%
        hostgroups      opsi-clients
        alias           %hostId%
        address         %hostId%
        }
'''

backend = BackendManager(
             dispatchConfigFile = u'/etc/opsi/backendManager/dispatch.conf',
             backendConfigDir   = u'/etc/opsi/backends',
             extensionConfigDir = u'/etc/opsi/backendManager/extend.d',
                       )


hosts = backend.host_getObjects(type="OpsiClient")

for host in hosts:
        filename = "%s.cfg" % host.id
        entry = template.replace("%hostId%",host.id)
        f = open(filename, 'w')
        f.write(entry)
        f.close()
```

20.5.7 opsi Check-Kommandos Konfiguration: `opsicommands.cfg`

Die oben beschriebenen Check-Kommandos müssen nun mit den bisherigen Kommandos konfiguriert werden. Dafür sollte eine Datei mit dem Namen `opsicommands.cfg` erstellt werden. Hier wird eine Bespielkonfiguration angegeben; je nach dem welche Funktionen Sie auf welche Weise verwenden wollen, müssen Sie diese Einstellungen nach eigenem Ermessen modifizieren.

Als erstes wird der Aufbau anhand eines Beispiels erläutert:

```
define command{
        command_name    check_opsi_clientstatus
        command_line            $USER1$/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p \
    $USER3$ -t checkClientStatus -c $HOSTADDRESS$
        }
```

Der `command_name` dient zur Referenzierung der weiteren Konfiguration. In der Option `command_line` werden die Konfigurationen und das Check-Kommando als erstes vereint.

Nach diesem Prinzip baut man nun die ganze Datei `opsicommands.cfg` auf:

```
define command {
        command_name    check_opsiwebservice
        command_line    /usr/lib/nagios/plugins/check_opsi -H $HOSTADDRESS$ -P 4447 -u $USER2$ -p $USER3$ -t \
    checkOpsiWebservice
}
define command {
        command_name    check_opsidiskusage
        command_line    /usr/lib/nagios/plugins/check_opsi -H $HOSTADDRESS$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p \
    $USER3$ -t checkOpsiDiskUsage
}
define command {
        command_name    check_opsiclientstatus
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
     -p $USER3$ -t checkClientStatus -c $HOSTADDRESS$
}
define command {
        command_name    check_opsiproductstatus
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
     -p $USER3$ -t checkProductStatus -e $ARG1$ -d $HOSTADDRESS$ -v
}
```

```
define command {
        command_name    check_opsiproductStatus_withGroups
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
      -p $USER3$ -t checkProductStatus -g $ARG1$ -G $ARG2$ -d "all"
}
define command {
        command_name    check_opsiproductStatus_withGroups_long
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
      -p $USER3$ -t checkProductStatus -g $ARG1$ -G $ARG2$ -v -d "all"
}
define command {
        command_name    check_opsidepotsync
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
      -p $USER3$ -t checkDepotSyncStatus -d $ARG1$
}
define command {
        command_name    check_opsidepotsync_long
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
      -p $USER3$ -t checkDepotSyncStatus -d $ARG1$ -v
}
define command {
        command_name    check_opsidepotsync_strict
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
      -p $USER3$ -t checkDepotSyncStatus -d $ARG1$ --strict
}
define command {
        command_name    check_opsidepotsync_strict_long
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
      -p $USER3$ -t checkDepotSyncStatus -d $ARG1$ --strict -v
}
define command {
        command_name    check_opsipluginon_client
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
      -p $USER3$ -t checkPluginOnClient -c $HOSTADDRESS$ --plugin $ARG1$
}
define command {
        command_name    check_opsipluginon_client_with_states
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTCONFIGSERVER$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$\
      -p $USER3$ -t checkPluginOnClient -c $HOSTADDRESS$ --plugin $ARG1$ -s $SERVICESTATEID$ -o "$SERVICEOUTPUT$"
}
define command {
        command_name    check_opsipluginon_client_from_depot
        command_line    /usr/lib/nagios/plugins/check_opsi -H $_HOSTDEPOTID$ -P $_HOSTCONFIGSERVERPORT$ -u $USER2$ -p \
     $USER3$ -t checkPluginOnClient -c $HOSTADDRESS$ --plugin $ARG1$
}
```

20.5.8 Kontakte: `opsicontacts.cfg`

```
define contact{
        contact_name                    adminuser
        alias                           Opsi
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c,r
        host_notification_options       d,r
        service_notification_commands   notify-service-by-email
        host_notification_commands      notify-host-by-email
        email                           root@localhost
        }
define contactgroup{
        contactgroup_name       opsiadmins
        alias                   Opsi Administrators
        members                 adminuser
        }
```

Hier müssen Sie natürlich *adminuser* in einen bzw. mehrere reale User abändern.

20.5.9 Services: `opsiservices.cfg`

In der Konfiguration der *Services* wird nun endlich angegeben, was der Nagios-Server monitoren und anzeigen soll. Dabei wird nun auf die bisher definierten Templates, commands und hostgroups bzw. hosts zugegriffen.

Zunächst der Teil der Services, die die tatsächlichen Zustände der Server betreffen. Hierbei wird beim Check auf den Depotsync gegen alle bekannten Services (*all*) geprüft.

```
#OPSI-Services
define service{
        use                             opsi-service-tmpl,srv-pnp
        hostgroup_name                  opsi-server
        service_description             opsi-webservice
        check_command                   check_opsiwebservice
        check_interval                  1
        }
define service{
        use                             opsi-service-tmpl
        hostgroup_name                  opsi-server
        service_description             opsi-diskusage
        check_command                   check_opsidiskusage
        check_interval                  1
        }
define service{
        use                             opsi-service-tmpl
        hostgroup_name                  opsi-server
        service_description             opsi-depotsyncstatus-longoutput
        check_command                   check_opsidepotsync_long!all
        check_interval                  10
        }
define service{
        use                             opsi-service-tmpl
        hostgroup_name                  opsi-server
        service_description             opsi-depotsyncstatus-strict-longoutput
        check_command                   check_opsidepotsync_strict_long!all
        check_interval                  10
        }
```

Nun der Teil der das Softwarerollout überwacht. Dabei wird in einem Check auf das opsi-Produkt *opsi-client-agent* verwiesen und zwei andere Checks verwenden hier eine opsi-Produktgruppe *opsiessentials* sowie eine opsi-Clientgruppe *productiveclients*.

```
define service{
        use                             opsi-service-tmpl
        hostgroup_name                  opsi-clients
        service_description             opsi-clientstatus
        check_command                   check_opsiclientstatus
        check_interval                  10
        }
define service{
        use                             opsi-service-tmpl
        hostgroup_name                  opsi-server
        service_description             opsi-productstatus-opsiclientagent
        check_command                   check_opsiproductstatus!opsi-client-agent
        check_interval                  10
        }
define service{
        use                             opsi-service-tmpl
        hostgroup_name                  opsi-server
        service_description             opsi-productstatus-opsiessentials-group
        check_command                   check_opsiproductStatus_withGroups!opsiessentials!productiveclients
        check_interval                  10
        }
define service{
        use                             opsi-service-tmpl
        hostgroup_name                  opsi-server
        service_description             opsi-productstatus-opsiessentials-group-longoutput
```

```
        check_command                   check_opsiproductStatus_withGroups_long!opsiessentials!productiveclients
        check_interval                  10
        }
```

Im dritten und letzten Teil werden die direkten Checks für die Clients definiert. Diese Checks richten sich in diesem Beispiel nicht an hostgroups, sondern einzelne hosts (*client.domain.local*,*depotclient.domain.local*).

- *opsi-direct-checkpluginonclient*
  Liefert einen normalen direkten Check vom Client und ein unknown, wenn der Client ausgeschaltet ist. Dabei wird versucht den Client direkt vom Configserver aus zu erreichen.
- *opsi-direct-checkpluginonclient-with-servicestate*
  verhält sich analog zu *opsi-direct-checkpluginonclient* doch wenn der Client ausgeschaltet ist, liefert er das letzte erfolgreiche Checkergebnis.
- *opsi-direct-checkpluginonclient-from-depot*
  verhält sich analog zu *opsi-direct-checkpluginonclient* nur wird versucht, den Client von der, in der Client Konfiguration angegeben, *_depotid* zu erreichen.

```
define service{
        use                             opsi-service-tmpl
        host_name                       client.domain.local,depotclient.domain.local
        service_description             opsi-direct-checkpluginonclient
        check_command                   check_opsipluginon_client!"C:\\opsi.org\\nagiosplugins\\check_memory.exe"
        check_interval                  10
        }
define service{
        use                             opsi-service-tmpl
        host_name                       client.domain.local
        service_description             opsi-direct-checkpluginonclient-with-servicestate
        check_command                   check_opsipluginon_client_with_states!"C:\\opsi.org\\nagiosplugins\\\
    check_memory.exe"
        check_interval                  10
        }
define service{
        use                             opsi-service-tmpl
        host_name                       depotclient.domain.local
        service_description             opsi-direct-checkpluginonclient-from-depot
        check_command                   check_opsipluginon_client_from_depot!"C:\\opsi.org\\nagiosplugins\\check_memory\
    .exe"
        check_interval                  10
        }
```

21 Datenhaltung von opsi (Backends)

21.1 file-Backend

Bei Verwendung des *file-Backends* liegen die Konfigurationsinformationen in Ini-Dateien auf dem Server.

Wesentliche Merkmale des Backends *file* :

- Aktuelles Defaultbackend von opsi
- Die Dateien dieses Backends liegen unter `/var/lib/opsi`.

Inhalt und Aufbau dieser Dateien ist im Kapitel Abschnitt 23.4 näher erläutert.

21.2 ldap-Backend

Die opsi 4 Backends sind in `/etc/opsi/backends/*.conf` konfiguriert. Bei Verwendung des *ldap-Backends* müssen hier die Daten eingetragen werden, die einen Zugriff auf den LDAP ermöglichen.

Weiterhin muss konfiguriert werden für welche Methoden auf das LDAP Backend zurückgegriffen wird. Lesen Sie dazu das Kapitel *Backendkonfiguration* des Getting Started Handbuchs.

Unterhalb der LDAP-Basis liegt eine *organizationalRole cn=opsi* (z.B. cn=opsi, dc=uib, dc=local). Unterhalb von opsi finden Sie die komplette LDAP-Datenhaltung. Diese läßt sich mit einem grafischen Frontend wie dem *Jxplorer* (z.B. aus den *opsi-adminutils*) leicht erkunden.

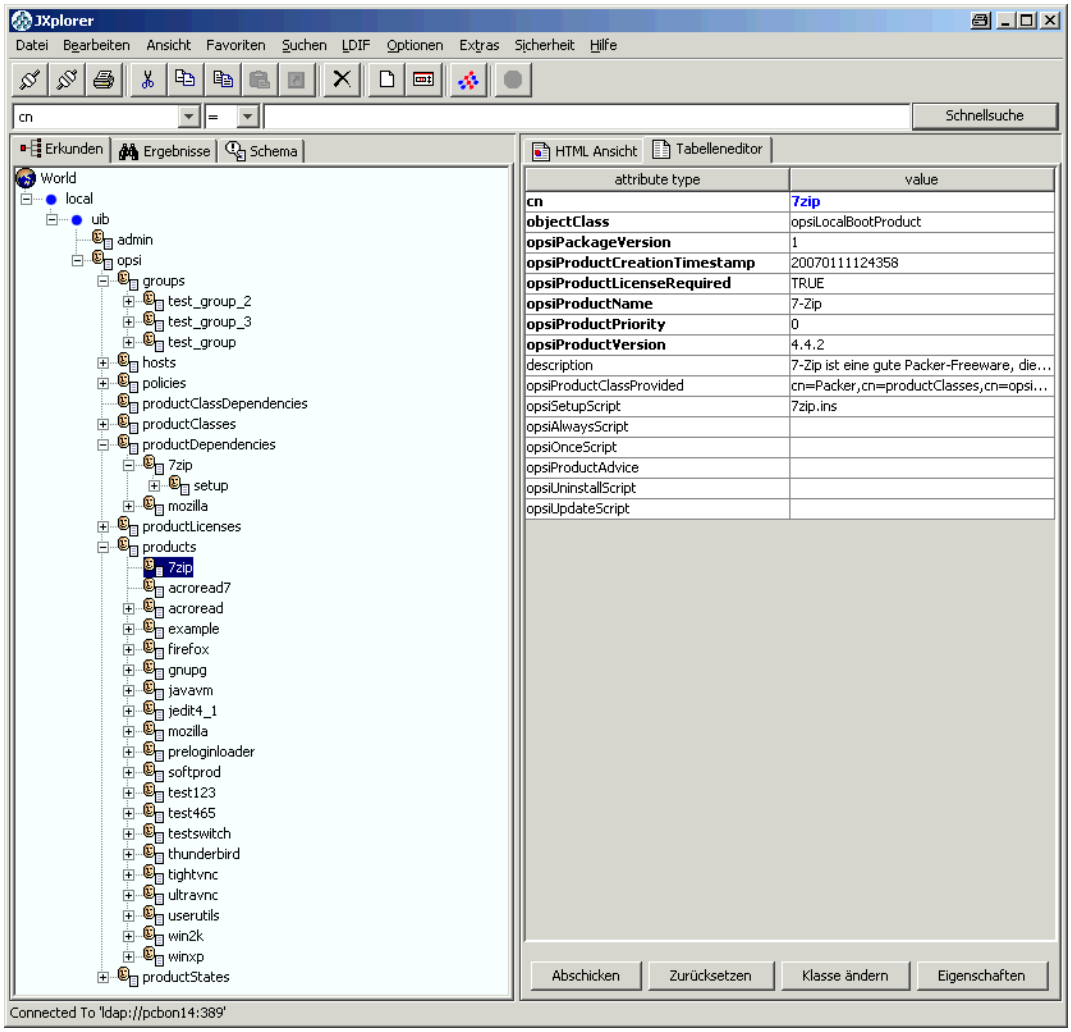


Abbildung 86: opsi ldap backend im Jxplorer

21.3 mysql-Backend

21.3.1 mysql-Backend für Inventarisierungsdaten (Übersicht und Datenstruktur)

Die Daten der Hardware- und Softwareinventarisierung werden per default über das opsi *file-Backend* in Textdateien abgelegt. Diese Form der Ablage ist für freie Abfragen und Reports weniger geeignet. Hierfür bietet sich die Ablage der Daten in einer SQL-Datenbank an.

Wesentliche Merkmale des Backends *mysql* :

- Optional (nicht das default Backend)
- Für Inventarisierungsdaten kostenfrei, für die Nutzung für sonstige Daten benötigen Sie eine kostenpflichtige Freischaltung.
- Fein granulierte Datenstruktur zur Datenhaltung und zusätzlich vereinfachtes Datenmodell für Abfragen.
- Eine Historyfunktion, welche Änderungen an den Inventarisierungsdaten protokolliert.

Bedingt durch die sehr unterschiedliche Natur der zu inventarisierenden Hardwarekomponenten ist die Datenstruktur in etwa wie folgt aufgebaut:

- Eine Tabelle *host* beschreibt alle bekannten Clients und stellt eine eindeutige *host_id* bereit.
- Für jeden Device-Typ gibt es zwei Tabellen:
  - *HARDWARE_DEVICE_ ....* beschreibt das Device z.B. Netzwerkkartentyp mit PCI-Kennung
  - *HARDWARE_CONFIG...* beschreibt Konfiguration der konkreten Netzwerkkarte z.B. MAC-Adresse. Die beiden Tabellen sind über das Feld *hardware_id* miteinander verbunden.

Ähnlich sieht es für die Softwareinventarisierung aus. Auch hier beschreibt die Tabelle *Software* die insgesamt gefundene Software während die Tabelle *Software_Config* die Client spezifische Konfiguration speichert.

Daraus ergibt sich folgende Liste von Tabellen:

```
HARDWARE_CONFIG_1394_CONTROLLER
HARDWARE_CONFIG_AUDIO_CONTROLLER
HARDWARE_CONFIG_BASE_BOARD
HARDWARE_CONFIG_BIOS
HARDWARE_CONFIG_CACHE_MEMORY
HARDWARE_CONFIG_COMPUTER_SYSTEM
HARDWARE_CONFIG_DISK_PARTITION
HARDWARE_CONFIG_FLOPPY_CONTROLLER
HARDWARE_CONFIG_FLOPPY_DRIVE
HARDWARE_CONFIG_HARDDISK_DRIVE
HARDWARE_CONFIG_IDE_CONTROLLER
HARDWARE_CONFIG_KEYBOARD
HARDWARE_CONFIG_MEMORY_BANK
HARDWARE_CONFIG_MEMORY_MODULE
HARDWARE_CONFIG_MONITOR
HARDWARE_CONFIG_NETWORK_CONTROLLER
HARDWARE_CONFIG_OPTICAL_DRIVE
HARDWARE_CONFIG_PCI_DEVICE
HARDWARE_CONFIG_PCMCIA_CONTROLLER
HARDWARE_CONFIG_POINTING_DEVICE
HARDWARE_CONFIG_PORT_CONNECTOR
HARDWARE_CONFIG_PRINTER
HARDWARE_CONFIG_PROCESSOR
HARDWARE_CONFIG_SCSI_CONTROLLER
HARDWARE_CONFIG_SYSTEM_SLOT
HARDWARE_CONFIG_TAPE_DRIVE
HARDWARE_CONFIG_USB_CONTROLLER
HARDWARE_CONFIG_VIDEO_CONTROLLER
HARDWARE_DEVICE_1394_CONTROLLER
HARDWARE_DEVICE_AUDIO_CONTROLLER
HARDWARE_DEVICE_BASE_BOARD
HARDWARE_DEVICE_BIOS
HARDWARE_DEVICE_CACHE_MEMORY
HARDWARE_DEVICE_COMPUTER_SYSTEM
HARDWARE_DEVICE_DISK_PARTITION
HARDWARE_DEVICE_FLOPPY_CONTROLLER
HARDWARE_DEVICE_FLOPPY_DRIVE
HARDWARE_DEVICE_HARDDISK_DRIVE
HARDWARE_DEVICE_IDE_CONTROLLER
HARDWARE_DEVICE_KEYBOARD
```

```
HARDWARE_DEVICE_MEMORY_BANK
HARDWARE_DEVICE_MEMORY_MODULE
HARDWARE_DEVICE_MONITOR
HARDWARE_DEVICE_NETWORK_CONTROLLER
HARDWARE_DEVICE_OPTICAL_DRIVE
HARDWARE_DEVICE_PCI_DEVICE
HARDWARE_DEVICE_PCMCIA_CONTROLLER
HARDWARE_DEVICE_POINTING_DEVICE
HARDWARE_DEVICE_PORT_CONNECTOR
HARDWARE_DEVICE_PRINTER
HARDWARE_DEVICE_PROCESSOR
HARDWARE_DEVICE_SCSI_CONTROLLER
HARDWARE_DEVICE_SYSTEM_SLOT
HARDWARE_DEVICE_TAPE_DRIVE
HARDWARE_DEVICE_USB_CONTROLLER
HARDWARE_DEVICE_VIDEO_CONTROLLER
HOST
SOFTWARE
SOFTWARE_CONFIG
```

Die Zuordnung der Spaltennamen zu einzelnen Deviceklassen ergibt sich aus folgender Liste (`/etc/opsi/hwaudit/locales/de_DE`):

```
DEVICE_ID.deviceType = Gerätetyp
DEVICE_ID.vendorId = Hersteller-ID
DEVICE_ID.deviceId = Geräte-ID
DEVICE_ID.subsystemVendorId = Subsystem-Hersteller-ID
DEVICE_ID.subsystemDeviceId = Subsystem-Geräte-ID
DEVICE_ID.revision= Revision
BASIC_INFO.name = Name
BASIC_INFO.description = Beschreibung
HARDWARE_DEVICE.vendor = Hersteller
HARDWARE_DEVICE.model = Modell
HARDWARE_DEVICE.serialNumber = Seriennummmer
COMPUTER_SYSTEM = Computer
COMPUTER_SYSTEM.systemType = Typ
COMPUTER_SYSTEM.totalPhysicalMemory = Arbeitsspeicher
BASE_BOARD = Hauptplatine
BASE_BOARD.product = Produkt
BIOS = BIOS
BIOS.version = Version
SYSTEM_SLOT = System-Steckplatz
SYSTEM_SLOT.currentUsage = Verwendung
SYSTEM_SLOT.status = Status
SYSTEM_SLOT.maxDataWidth = Max. Busbreite
PORT_CONNECTOR = Port
PORT_CONNECTOR.connectorType = Attribute
PORT_CONNECTOR.internalDesignator = Interne Bezeichnung
PORT_CONNECTOR.internalConnectorType = Interner Typ
PORT_CONNECTOR.externalDesignator = Externe Bezeichnung
PORT_CONNECTOR.externalConnectorType = Externer Typ
PROCESSOR = Prozessor
PROCESSOR.architecture = Architektur
PROCESSOR.family = Familie
PROCESSOR.currentClockSpeed = Momentane Taktung
PROCESSOR.maxClockSpeed = Maximale Taktung
PROCESSOR.extClock = Externe Taktung
PROCESSOR.processorId = Prozessor-ID
PROCESSOR.addressWidth = Adress-Bits
PROCESSOR.socketDesignation = Zugehöriger Sockel
PROCESSOR.voltage = Spannung
MEMORY_BANK = Speicher-Bank
MEMORY_BANK.location = Position
MEMORY_BANK.maxCapacity = Maximale Kapazität
MEMORY_BANK.slots = Steckplätze
MEMORY_MODULE = Speicher-Modul
MEMORY_MODULE.deviceLocator = Zugehöriger Sockel
MEMORY_MODULE.capacity = Kapazität
```

```
MEMORY_MODULE.formFactor = Bauart
MEMORY_MODULE.speed = Taktung
MEMORY_MODULE.memoryType = Speichertyp
MEMORY_MODULE.dataWidth = Datenbreite
MEMORY_MODULE.tag = Bezeichnung
CACHE_MEMORY = Zwischenspeicher
CACHE_MEMORY.installedSize = Installierte Größe
CACHE_MEMORY.maxSize = Maximale Größe
CACHE_MEMORY.location = Position
CACHE_MEMORY.level = Level
PCI_DEVICE = PCI-Gerät
PCI_DEVICE.busId = Bus-ID
NETWORK_CONTROLLER = Netzwerkkarte
NETWORK_CONTROLLER.adapterType = Adapter-Typ
NETWORK_CONTROLLER.maxSpeed = Maximale Geschwindigkeit
NETWORK_CONTROLLER.macAddress = MAC-Adresse
NETWORK_CONTROLLER.netConnectionStatus = Verbindungsstatus
NETWORK_CONTROLLER.autoSense = auto-sense
AUDIO_CONTROLLER = Audiokarte
IDE_CONTROLLER = IDE-Controller
SCSI_CONTROLLER = SCSI-Controller
FLOPPY_CONTROLLER = Floppy-Controller
USB_CONTROLLER = USB-Controller
1394_CONTROLLER = 1394-Controller
PCMCIA_CONTROLLER = PCMCIA-Controller
VIDEO_CONTROLLER = Grafikkarte
VIDEO_CONTROLLER.videoProcessor = Video-Prozessor
VIDEO_CONTROLLER.adapterRAM = Video-Speicher
DRIVE.size = Größe
FLOPPY_DRIVE = Floppylaufwerk
TAPE_DRIVE = Bandlaufwerk
HARDDISK_DRIVE = Festplatte
HARDDISK_DRIVE.cylinders = Cylinder
HARDDISK_DRIVE.heads = Heads
HARDDISK_DRIVE.sectors = Sektoren
HARDDISK_DRIVE.partitions = Partitionen
DISK_PARTITION = Partition
DISK_PARTITION.size = Größe
DISK_PARTITION.startingOffset = Start-Offset
DISK_PARTITION.index = Index
DISK_PARTITION.filesystem = Dateisystem
DISK_PARTITION.freeSpace = Freier Speicher
DISK_PARTITION.driveLetter = Laufwerksbuchstabe
OPTICAL_DRIVE = Optisches Laufwerk
OPTICAL_DRIVE.driveLetter = Laufwerksbuchstabe
MONITOR = Monitor
MONITOR.screenHeight = Vertikale Auflösung
MONITOR.screenWidth = Horizontale Auflösung
KEYBOARD = Tastatur
KEYBOARD.numberOfFunctionKeys = Anzahl Funktionstasten
POINTING_DEVICE = Zeigegerät
POINTING_DEVICE.numberOfButtons = Anzahl der Tasten
PRINTER = Drucker
PRINTER.horizontalResolution = Vertikale Auflösung
PRINTER.verticalResolution = Horizontale Auflösung
PRINTER.capabilities = Fähigkeiten
PRINTER.paperSizesSupported = Unterstützte Papierformate
PRINTER.driverName = Name des Treibers
PRINTER.port = Anschluss
```

Beispiele für Abfragen: Liste aller Festplatten:

```
SELECT * FROM HARDWARE_DEVICE_HARDDISK_DRIVE D
LEFT OUTER JOIN HARDWARE_CONFIG_HARDDISK_DRIVE H ON D.hardware_id=H.hardware_id ;
```

Die Softwareinventarisierung verwendet als Hauptschlüssel die folgenden Felder:

- Name
  Dieser ist der *windowsDisplayName* bzw. wenn dieser nicht vorhanden ist die *windowsSoftwareId*. Beide werden aus der Registry ermittelt:
  HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall bzw.
  HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\<id> DisplayName
- Version
  HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\<id> DisplayVersion
- SubVersion
- Language
- Architecture (32 Bit / 64 Bit)

In der Tabelle *Software_config* sind diese Felder zum Feld *config_id* zusammengefasst.

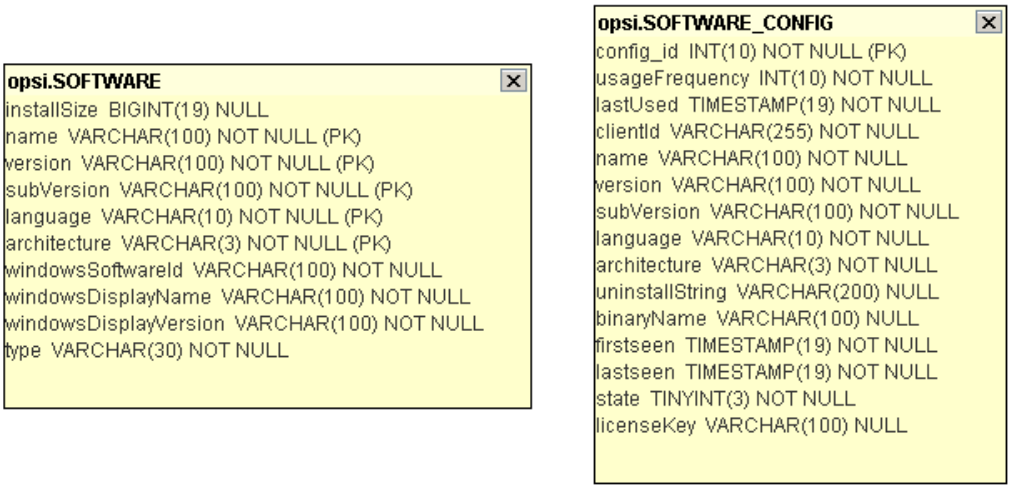


Abbildung 87: Datenbankschema: Softwareinventarisierung

21.3.2 mysql-Backend für Konfigurationsdaten (Übersicht)

Das *mysql-Backend* für Konfigurationsdaten steht seit opsi 4.0 zur Verfügung.

Dieses Modul ist momentan eine kofinanzierte opsi Erweiterung. Das bedeutet die Verwendung ist nicht kostenlos. Weitere Details hierzu finden Sie in Abschnitt 6.

Das *mysql-Backend* hat den Vorteil der höheren Performanz insbesondere bei großen Installationen.

Hier eine Übersicht über die Datenstruktur:

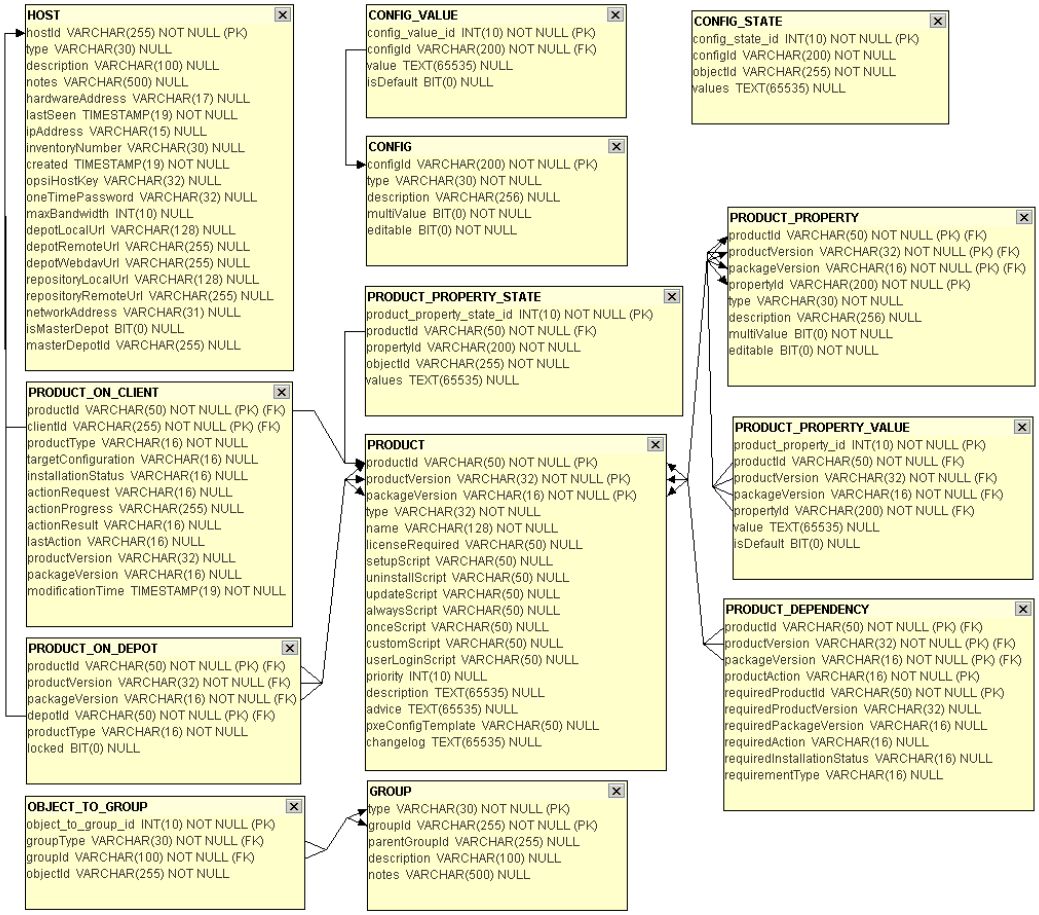


Abbildung 88: Datenbankschema: Konfigurationsdaten

21.3.3 Initialisierung des mysql-Backends

Wenn der mysql-server noch nicht installiert ist, muss dies zunächst erfolgen mit:

```
apt-get install mysql-server
```

Danach muss für der root Zugang von mysql ein Passwort gesetzt werden:

```
mysqladmin --user=root password linux123
```

Mit dem Befehl `opsi-setup --configure-mysql` kann nun die Datenbank aufgebaut werden.

Eine Beispiel-Sitzung:

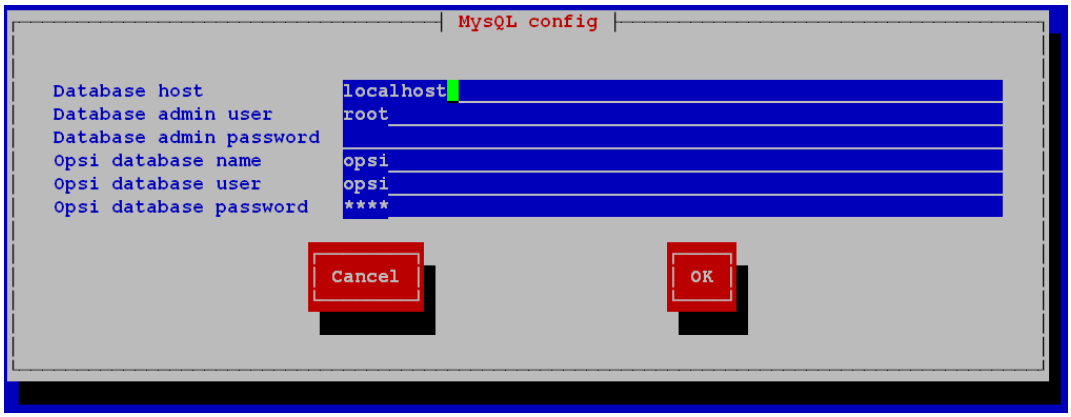


Abbildung 89: opsi-setup --configure-mysql: Eingabemaske

```
Connecting to host 'localhost' as user 'root'
Successfully connected to host 'localhost' as user 'root'
Creating database 'opsi'
Database 'opsi' created
Creating user 'opsi' and granting all rights on 'opsi'
User 'opsi' created and privileges set
Testing connection to database 'opsi' as user 'opsi'
Successfully connected to host 'localhost' as user 'opsi'
Updating backend config '/etc/opsi/backends/mysql.conf'
Backend config '/etc/opsi/backends/mysql.conf' updated
Initializing mysql backend
```

Abbildung 90: opsi-setup --configure-mysql: Ausgabe

Bei den Abfragen können außer beim Passwort alle Vorgaben mit Enter bestätigt werden.

Als nächstes muss in der `/etc/opsi/backendManager/dispatch.conf` eingetragen werden, das das *mysql-Backend* auch verwendet werden soll. Eine genaue Beschreibung zu dieser Konfiguration finden Sie im Kapitel *Backend-Konfiguration* des *getting-started Handbuchs*. Die Datei selbst enthält eine Reihe von Beispielen typischer Konfigurationen. Eine Konfiguration für *mysql-Backend* (ohne internen DHCPD) sieht so aus:

```
backend_.*         : mysql, opsipxeconfd
host_.*            : mysql, opsipxeconfd
productOnClient_.* : mysql, opsipxeconfd
configState_.*     : mysql, opsipxeconfd
.*                 : mysql
```

Nach Abschluss dieser Konfigurationsarbeit müssen Sie den folgenden Befehlen die Benutzung der jetzt konfigurierten und konvertierten Backend aktivieren:

```
opsi-setup --init-current-config
opsi-setup --set-rights
/etc/init.d/opsiconfd restart
/etc/init.d/opsipxeconfd restart
```

21.3.4 Konfigurieren der MySQL-Datenbank zum Zugriff von außen

has to be written

21.4 HostControl-Backend

Das HostControl-Backend speichert keine Konfigurationsdaten, sondern dient der Steuerung von opsi-Clients. Hierzu gehören beispielsweise das Starten von Clients per Wake-On-LAN oder das Senden von Steuerungsbefehlen an den opsi-client-agent.

Die Konfiguration des HostControl-Backends wird in der Konfigurationsdatei `/etc/opsi/backends/hostcontrol.conf` vorgenommen. Konfigurations-Optionen sind hierbei:

- `opsiclientdPort`:
  Netzwerk-Port für die Verbindungsaufnahme zu einem opsi-client-agent.
- `hostRpcTimeout`:
  Timeout (in Sekunden) bei der Verbindungsaufnahme zu einem opsi-client-agent.
- `resolveHostAddress`:
  Steht diese Option auf `True`, wird bei einem Verbindungsaufbau vom opsi-server zu einem opsi-client die IP-Adresse des Clients bevorzugt über die Namensauflösung ermittelt. Um die im Backend von opsi hinterlegte IP-Adresse zu bevorzugen ist die Option auf `False` zu setzen.
- `maxConnections`:
  Maximale Anzahl simultaner Verbindungen zu opsi-client-agents.
- `broadcastAddresses`:
  Liste von Broadcast-Adressen für das Versenden von Wake-On-LAN-Broadcasts.

21.5 Konvertierung zwischen Backends

Der Befehl `opsi-convert` dient zum Konvertieren der opsi-Konfigurationsdaten zwischen verschiedenen Backends. Das Ziel oder Quelle kann auf verschieden Arten bestimmt werden:

- Backendnamen:
  Durch Angabe des Namen wird ein entsprechendes Backend auf dem aktuellen Server angegeben. So konvertiert `opsi-convert file mysql` auf dem aktuellen Server vom *file-Backend* zum *mysql-Backend*.
- Service-Adresse
  Durch Angaben von Serviceadressen kann ein Server z.B. auch Remote angesprochen werden. Die Service Adresse hat die Form
  https://<username>@<ipadresse>:4447/rpc Nach den Passwörtern wird gefragt.
  Beispiel:

```
opsi-convert -s -l /tmp/log https://uib@192.168.2.162:4447/rpc \   https://opsi@192.168.2.42:4447/rpc
```

- Konfigurationsverzeichnis
  Durch Angabe von Konfigurationsverzeichnissen für die entsprechende Backendmanagerkonfiguration können Quelle bzw. Ziel sehr detailliert beschrieben werden.

```
opsi-convert --help

Usage: opsi-convert [options] <from> <to>
Convert an opsi database into an other.
Options:
    -h        show this help text
    -V        show version information
    -q        do not show progress
    -v        increase verbosity (can be used multiple times)
    -c        clean destination database before writing
    -s        use destination host as new server
    -l <file> log to this file
```

```
<from> and <to> can be:
     - the name of a backend as defined in /etc/opsi/backends (file, ldap, ...)
     - the url of a opsi configuration service
       http(s)://<user>@<host>:<port>/rpc
```

21.6 Bootdateien

Unter `/tftpboot/linux` finden sich die Bootdateien, die im Zusammenspiel mit den PXE-Bootproms benötigt werden.

21.7 Absicherung der Shares über verschlüsselte Passwörter

Der *opsi-client-agent* greift auf die vom *opsi-server* zur Verfügung gestellten Shares zu, um die dort liegende Software zu installieren zu können.

Hierzu wird der System-User *pcpatch* verwendet. Die Absicherung dieser Shares und damit der Authentifizierungsdaten des Users pcpatch sind wichtig für die: * allgemeine Systemsicherheit und Datenintegrität * Absicherung der potenziell lizenzpflichtigen Softwarepakete gegen missbräuchliche Nutzung

Um dem *opsi-client-agent* ein Zugriff auf die Authentifizierungsdaten zu ermöglichen, wird für jeden Client bei seiner Erzeugung in opsi ein spezifischer Schlüssel (*opsi-host-Schlüssel*) erzeugt. Dieser Schlüssel wird zum einen (beim *file-Backend*) in der Datei `/etc/opsi/pckeys` abgelegt und zum anderen dem PC bei der Reinstallation übergeben. Der übergebene Schlüssel wird im Rahmen der der Installation des *opsi-client-agent* in der Datei `c:\program files\opsi.org\opsi-client-agent\opsiclientd\opsiclientd.conf` so abgelegt, dass nur Administratoren Zugriff darauf haben. Ebenso hat auf dem *opsi-server* nur root und Mitglieder der Gruppe *opsiadmin* Zugriff auf die Datei `/etc/opsi/pckeys`. Auf diese Weise verfügt jeder PC über einen Schlüssel, der nur dem PC und dem *opsi-server* bekannt ist und der gegenüber dem Zugriff durch normale Anwender geschützt ist. Mit diesem Schlüssel wird das aktuelle Passwort des system users *pcpatch* auf dem *opsi-server* verschlüsselt und im Backend abgelegt. Dieses verschlüsselte Passwort wird vom Client bei jeder Aktivierung des *opsi-client-agent* neu gelesen, so dass eine Änderung des *pcpatch* Passwortes jederzeit möglich ist und der Client auf verschlüsseltem Wege das veränderte Passwort erfährt.

22 Anpassen des opsi-client-agent an Corporate Identity (CI)

Die Anpassung des Erscheinungsbildes des *opsi-client-agent* kann insbesondere bei der Einführung erheblich zur Akzeptanz beizutragen und eine mögliche Verunsicherung der Anwender durch das Auftreten von bisher ungewohnten Fenstern zu vermeiden.

Der *opsi-winst* lässt sich über eine Manipulation der Grafik `bg.png` im Unterverzeichnis `winstskin` frei anpassen. Nach einer Veränderung der `bg.png` unter `/opt/pcbin/install/opsi-client-agent/files/opsi/opsi-winst/winstskin` setzen Sie mit dem Befehl

```
touch /opt/pcbin/install/opsi-client-agent/files/opsi/opsi-winst/winst32.exe
```

das Datum des *opsi-winst* neu. In der Folge wird beim nächsten Start des *opsi-client-agent* der *opsi-winst* mit der veränderten `bg.png` automatisch auf den Client geladen.

Weiterhin finden Sie im Verzeichnis `/opt/pcbin/install/opsi-client-agent/files/opsi/dist/notifier` die Dateien `action.bmp` und `event.bmp` welche Sie auch an Ihre Bedürfnisse anpassen können. Diese Dateien werden bei einer Installation des *opsi-client-agent* übertragen.

Diese Dateien werden beim Einspielen von Updates des Produktes *opsi-client-agent* nicht überschrieben wenn Sie diese verändert haben.

Weiterhin können Sie die Orte, Farbe, Schrift an denen Text in diese Grafiken eingeblendet wird über die entsprechenden Ini-Dateien (`action.ini, event.ini, shutdown.ini`) ebenfalls anpassen. (CAUTION: Diese Dateien sind im Moment nicht gegen ein Update geschützt).

Die erforderlichen Anpassungen lassen sich auch in einem *opsi-winst*-script durchführen, welches als eigenes Produkt oder als *Update*-Script in den *opsi-client-agent* eingebunden wird.

Hier ein Beispiel:

```
[Actions]
requiredWinstVersion >= "4.10.8.0"
Message "opsi-client-agent update"
ShowBitmap "%scriptpath%\opsi.png" "opsi-client-agent"


DefVar $INST_BaseDir$
DefVar $INST_NotifierDir$
DefVar $INST_OpsiclientdDir$
DefVar $INST_OpsiclientdConf$
DefVar $INST_WinstDir$
DefVar $OLB_LoginBlockerStart$
DefVar $block_login$
DefVar $winst_skin_color$
DefVar $action_color$
DefVar $count_event_abort$
DefVar $warning_before_event_sec$

Set $INST_BaseDir$    = "%ProgramFilesDir%\opsi.org\opsi-client-agent"
Set $INST_OpsiclientdDir$    = $INST_BaseDir$+"\opsiclientd"
Set $INST_OpsiclientdConf$ = $INST_OpsiclientdDir$+"\opsiclientd.conf"
Set $INST_WinstDir$    = $INST_BaseDir$+"\opsi-winst"
Set $INST_NotifierDir$ = $INST_BaseDir$+"\notifier"
;set $winst_skin_color$ = "$00E7E7E7"
set $winst_skin_color$ = "$00202020"
set $action_color$ = "32,32,32"
;Set $OLB_LoginBlockerStart$                = GetProductProperty ("LoginBlockerStart","off")
set $count_event_abort$ = "2"
set $warning_before_event_sec$ = "20"

;if $OLB_LoginBlockerStart$ = "on"
;       set $block_login$ = "true"
;else
        set $block_login$ = "false"
;endif

;       if FileExists("%ScriptPath%\custom.ins")
;               comment "Start uninstall sub section"
;               Sub "%ScriptPath%\custom.ins"
;       endif

Files_copy_images
Patches_action_ini $INST_NotifierDir$+"\action.ini"
Patches_event_ini $INST_NotifierDir$+"\event.ini"
Patches_shutdown_ini $INST_NotifierDir$+"\shutdown.ini"
Patches_winst_skin $INST_WinstDir$+"\winstskin\skin.ini"
Patches_opsiclientd_conf $INST_OpsiclientdConf$

[Files_copy_images]
copy "%SCRIPTPATH%\notifier\*.*" "$INST_NotifierDir$"
copy "%SCRIPTPATH%\winstskin\*.*" "$INST_WinstDir$\winstskin"

[Patches_action_ini]
```

```
set [Form] Hidden = false
delsec [ProgressBarCurrentProgress]
delsec [ProgressBarOverallProgress]
set [LabelTitle] FontColor = $action_color$
set [LabelOpsiclientdInfo] FontColor = $action_color$
set [LabelActionProcessorInfo] FontColor = $action_color$
set [LabelStatus] FontColor = $action_color$
set [LabelDetail] FontColor = $action_color$
set [LabelConfigServiceUrl] FontColor = $action_color$
set [LabelClientId] FontColor = $action_color$

[Patches_event_ini]
set [LabelMessage] FontSize = 12
set [LabelMessage] FontBold = true
set [LabelMessage] Top = 95
set [LabelMessage] FontBold = true
set [LabelStatus] Left = 25
set [LabelStatus] Top = 55

[Patches_shutdown_ini]
set [LabelMessage] FontSize = 12
set [LabelMessage] FontBold = true
set [LabelMessage] Top = 95
set [LabelMessage] FontBold = true
set [LabelMessage] Left = 25
set [LabelMessage] Top = 55

[Patches_opsiclientd_conf]
set [event_gui_startup] message[de] = Darf jetzt Software / Updates installiert werden ?  Sie
    können diese Aktion insgesamt %user_cancelable% mal abbrechen. Die Aktion wurde bereits %
    cancel_counter% mal abgebrochen.
set [event_gui_startup] action_notifier_desktop = winlogon
set [event_gui_startup] action_processor_desktop = winlogon
set [event_gui_startup] user_cancelable = $count_event_abort$
set [event_gui_startup] block_login = $block_login$
set [event_gui_startup] warning_time = $warning_before_event_sec$
set [event_gui_startup] shutdown_user_cancelable = 2
set [event_gui_startup] shutdown_warning_message[de] = Ein Neustart wird benötigt um die Software
    -Installationen abzuschliessen. Sie können diesen Neustart insgesamt %
    shutdown_user_cancelable% mal verschieben. Der Neustart wurde bereits %
    shutdown_cancel_counter% mal verschoben.
set [event_gui_startup] shutdown_warning_repetition_time = 3600
set [event_gui_startup] shutdown_warning_time = 300
set [event_gui_startup] shutdown_notifier_command = %opsiclientd_notifier.command% -s notifier\\
    shutdown.ini
set [event_gui_startup] shutdown_notifier_desktop = current


[Patches_winst_skin]
set [LabelVersion] FontColor = $winst_skin_color$
set [LabelProduct] FontColor = $winst_skin_color$
set [LabelProduct] FontSize = 22
set [LabelInfo] FontColor = $winst_skin_color$
set [LabelDetail] FontColor = $winst_skin_color$
set [LabelCommand] FontColor = $winst_skin_color$
set [LabelProgress] FontColor = $winst_skin_color$
set [ProgressBar] BarColor = $winst_skin_color$
```

```
set [ProgressBar] StartColor = $winst_skin_color$
set [ProgressBar] FinalColor = $winst_skin_color$
set [ProgressBar] ShapeColor = $winst_skin_color$
;set [Form] Color = $00FFB359
set [Form] Color = $00E7E7E7
```

23 Wichtige Dateien des opsi-servers

23.1 Allgemeine Konfigurationsdateien in /etc

23.1.1 /etc/hosts

Hier können IP-Nummer und IP-Name der Clients eingetragen werden (zusätzliche Namen sind Aliase, ab dem Zeichen „#“ ist der Eintrag Kommentar).

opsi hätte gerne den *full qualified hostname* (also inclusive Domain) und dieser kann auch statt aus der `/etc/hosts` aus dem DNS kommen.

Beispiel:

```
192.168.2.106  dplaptop.uib.local  dplaptop  # this opsi-server
192.168.2.153  schleppi.uib.local
192.168.2.178  test_pc1.uib.local # Test-PC PXE-bootprom
```

Die Ausgabe von:

```
getent hosts $(hostname -f)
```

Das Ergebnis sollte beispielsweise so aussehen:

```
192.168.1.1 server.domain.tld server
```

Sieht das Ergebnis nicht so aus (enthält z.B. *127.0.0.1* oder *localhost*), dann müssen Sie Ihre `/etc/hosts` oder Namensauflösung zunächst korrigieren.

23.1.2 /etc/group

Hier müssen zwei Gruppen angelegt sein: pcpatch und opsidamin. In der Gruppe pcpatch sollten alle user sein, die mit Paketverwaltung zu tun haben. In der Gruppe opsiadmin müssen alle user sein, die den opsiconfd-Webservice verwenden wollen z.B. über den opsi-configed (oder das Applet).

23.1.3 /etc/opsi/backends/

Konfigurationsdateien der verwendeten Backends.

23.1.4 /etc/opsi/backendManager/

- `acl.conf`
  Konfiguration der Zugriffsrechte auf die opsi Methoden.
- `dispatch.conf`
  Konfiguration welche der unter `/etc/opsi/backends/` konfigurierten Backends wofür verwendet werden sollen.
- `extend.d/`
  Verzeichnis der Backenderweiterungen. So liegen hier z.B. die Scripte, welche die opsi 3 Methoden auf die opsi 4 Methoden mappen.

23.1.5 /etc/opsi/hwaudit/*

Ab Version 3.2

Hier finden sich Konfigurationen zur Hardwareinventarisierung.

Im Verzeichnis locales liegen die Sprachanpassungen.

In der Datei opsihwaudit.conf ist die Abbildung zwischen WMI Klassen und der opsi Datenhaltung konfiguriert.

23.1.6 /etc/opsi/opsi.conf

Seit Version 4.0.2-2

Allgemeine opsi Konfigurationen.

Beispiel:

```
[groups]
fileadmingroup = pcpatch
```

Hintergrund: Die klassischen Installationsvariante mit dem Benutzer: `pcpatch` mit der primären Gruppe: `pcpatch` funktioniert nicht mit Samba 4. Da Samba4 den grundlegenden Restriktionen von Active-Directory unterliegt, sind Gruppen mit der gleichen Bezeichnung wie User (wie in Unix/Linux üblich) nicht mehr erlaubt. Aus diesem Grund wurde eine neue Konfigurationsdatei eingeführt: `/etc/opsi/opsi.conf`, über die gesteuert wird, wie die Gruppe für den Samba-Zugriff auf die Freigaben bestimmt wird. So wird z.B. bei UCS 3 nun über diese Datei der Gruppenname `pcpatch` umbenannt und heißt von nun an: `opsifileadmins`. Das bedeutet, dass die User, die Zugriffsrechte für die Freigaben von opsi erhalten müssen (opsi-Paketierer) unter UCS 3 nicht Mitglied der Gruppe `pcpatch` werden können, sondern Mitglied der Gruppe `opsifileadmins` sein müssen. Diese Besonderheit gilt vorerst nur für UCS 3 und unterscheidet sich von den anderen Distributionen (solange bis diese auch Samba 4 einführen, oder Sie dort Samba 4 installieren)

23.1.7 /etc/opsi/modules

Ab Version 3.4

Von der uib gmbh signierte Datei zur Freischaltung kostenpflichtiger Features. Wird die Datei verändert, verliert sie Ihre Gültigkeit. Ohne diese Datei stehen nur die kostenlosen Features zur Verfügung.

23.1.8 /etc/opsi/opsiconfd.conf

Ab Version 3

Konfigurationsdatei für den opsiconfd in dem sonstige Konfigurationen wie Ports, Interfaces, Logging hinterlegt sind.

23.1.9 /etc/opsi/opsiconfd.pem

Ab Version 3

Konfigurationsdatei für den opsiconfd in dem das ssl-Zertifikat hinterlegt ist.

23.1.10 /etc/opsi/opsipxeconfd.conf

Konfigurationsdatei für den opsipxeconfd, der für das Schreiben der Startdateien für das Linux-Bootimage zuständig ist. Hier können Verzeichnisse, Defaults und Loglevel konfiguriert werden.

23.1.11 /etc/opsi/opsi-product-updater.conf

Konfigurationsdatei für den opsi-product-updater. Siehe Abschnitt 4.5

23.1.12 /etc/opsi/version

Enthält die Versionsnummer des installierten opsi.

23.1.13 /etc/init.d/

Start-Stop Skripte für: * opsi-atftpd * opsiconfd * opsipxeconfd

23.2 Bootdateien

23.2.1 Bootdateien in /tftpboot/linux

- `pxelinux.0`
  Bootfile, der im ersten Schritt vom PXE-Bootprom geladen wird.
- `install` und `miniroot.gz`
  Installationsbootimage, das per tftp an den Client bei der Reinstallation übertragen wird.

23.2.2 Bootdateien in /tftpboot/linux/pxelinux.cfg

- `01-<mac adresse>` bzw. `<IP-NUMMER-in-Hex>`
  Dateien mit der Hardwareadresse des Clients und dem Prefix 01 - sind auf dem *opsi-server* als clientspezifische Bootfiles zu finden. Sie sind zumeist über den *opsipxeconfd* als named pipes erzeugt und sollen eine Reinstallation des Clients einleiten.
- `default`
  Die Datei default wird geladen, wenn es keine clientspezifischen Dateien gibt. Wird diese Datei geladen, so bootet der Client danach lokal weiter.
- `install`
  Informationen zum boot des Installationsbootimages, die vom *opsipxeconfd* in die named pipe geschrieben werden.

23.3 Dateien in /var/lib/opsi

23.3.1 /var/lib/opsi/repository

Hier werden *Produkt-Pakete* gespeichert, welche über den Aufruf des `opsi-product-updater` auf den Server geladen werden.

Weiterhin werden hier *Produkt-Pakete* gespeichert, welche über den Aufruf des `opsi-package-manager` installiert werden, wenn dieser mit der Option `-d` aufgerufen wird.

23.3.2 /var/lib/opsi/depot

Dieses Verzeichnis ist (read-only) als Samba share *opsi_depot* freigegeben. Es dient unter der Distribution SLES direkt als Depot Verzeichnis (statt `/opt/pcbin/install`). Bei anderen Distributionen wird hier ein Link nach `/opt/pcbin/install` erzeugt. In Zukunft wird aus Gründen der LSB Konformität auch bei anderen Distributionen das Depotverzeichnis hier liegen.

23.3.3 /var/lib/opsi/ntfs-images

In diesem Verzeichnis werden (per default) Partionsimages abgelegt, welche mit dem Netboot-Produkt *ntfs-write-image* ausgelesen werden.

23.3.4 Weitere Verzeichnisse

Die restlichen Verzeichnisse in `/var/lib/opsi` (`config` und `audit`) sind Verzeichnisse des *file-Backends*, welche im folgenden Kapitel beschrieben sind.

23.4 Dateien des file Backends

23.4.1 /etc/opsi/pckeys

Hier sind die clientspezifischen *opsi-host-Schlüssels* sowie der Schlüssel des Servers selber abgelegt.

Beispiel:

```
schleppi.uib.local:fdc2493ace4b372fd39dbba3fcd62182
laptop.uib.local:c397c280fc2d3db81d39b4a4329b5f65
pcbon13.uib.local:61149ef590469f765a1be6cfbacbf491
```

23.4.2 /etc/opsi/passwd

Hier sind die mit dem Schlüssel des Servers verschlüsselten Passwörter (z.B. für pcpatch) abgelegt.

23.4.3 Übersicht /var/lib/opsi

Die Dateien des file Backends von opsi 4 finden sich standardmäßig in `/var/lib/opsi/config/`. Das folgende Schema gibt einen Überblick der Verzeichnisstruktur:

```
/var/lib/opsi-|
              |-depot                          opsi_depot share
              |-repository                     opsi package repository used by opsi-product-updater opsi-package-\
    manager
              |-audit                          inventory - files
              !-config/-|                              config share
                        |-clientgroups.ini     client groups
                        |-config.ini           Host Parameter (Global Defaults)
                        |-clients/             <pcname.ini> files
                        |-products/            product control files
                        !-depots               depot description files

       +audit/
               global.<Type> (Allgemeine Hard-, bzw. Softwareinformationen)
               <FQDN>.<Type> (Hard-, bzw. Softwareinformationen der Clients)

       clientgroups.ini (enthält HostGroups)

       +clients/
               <FQDN>.ini (Informationen der Clients)
       config.ini (enthält Configs)

       +depots/
               <FQDN>.ini (Informationen der Server)

       +products/
               <ID>_<ProdVer>-<PackVer>.<Type> (Informationen der Products)

       +templates/
               pcproto.ini (Vorlage für Clients)
               <FQDN>.ini (Vorlage für spezifische Clients)
```

23.4.4 Konfigurationsdateien im Detail

./clientgroups.ini

Die Datei enthält die Informationen über Client-Gruppen.

```
[<GroupId>]
<HostId> = 1 #aktiv
<HostId> = 0 #inaktiv
```

./config.ini

Hier finden sich die Defaultwerte der Serverkonfiguration wie im *opsi-configed* im Tab *Host Parameter* angezeigt.

./clients/<FQDN>.ini

In der dieser Datei werden die Client spezifischen Konfigurationen zusammen gefasst. Die Informationen werden mit denen aus der `<depot-id>.ini` zusammengefasst, wobei Informationen aus der `<FQDN>.ini` Vorrang haben.

Diese Dateien sind folgendermaßen aufgebaut:

Die Sektion *info* enthält alle direkt auf den Client bezogene Informationen, wie z.B. die Beschreibung.

```
[info]
description = <String>
created = <Date> #format: 'YYYY-MM-DD HH:MM:SS'
lastseen = <Date> #format: 'YYYY-MM-DD HH:MM:SS'
inventorynumber = <String>
notes = <String>
hardwareaddress = <MAC> #format: 'hh:hh:hh:hh:hh:hh'
ipaddress = <IP> #format: 'nnn.nnn.nnn.nnn'
onetimepassword = <String>
```

Die folgende Sektion beschreibt die aktuellen Zustände der Produkte auf dem Client. Wenn keine Einträge vorhanden sind, wird *not_installed:none* angenommen.

```
[<Type>_product_states] #'Local-', bzw. 'NetbootProduct'
<ProductId> = <InstallationStatus>:<ActionRequest>
```

Genauere Informationen stehen dazu in den, zu den jeweiligen Produkten zugehörigen, Sektionen.

```
[<ProductId>-state]
producttype = <Type> #'Local-', bzw. 'NetbootProduct'
actionprogress = <String>
productversion = <ProdVer>
packageversion = <PackVer>
modificationtime = <Date> #format: 'YYYY-MM-DD HH:MM:SS'
lastaction = <ActionRequest>
actionresult = <ActionResult>
targetconfiguration = <InstallationStatus>
```

/var/lib/opsi/config/templates

Hier findet sich die Datei `pcproto.ini`, welche das Standardtemplate zur Erzeugung neuer Client-Ini-Dateien ist und besitzt dieselbe Struktur. Wenn bestimmte Clients abweichende Informationen erhalten sollen, kann man auch jeweils eine `<FQDN>.ini` in diesem Verzeichnis ablegen.

/var/lib/opsi/config/depots/

Hier findet sich die Dateien der *opsi-depotserver*, die ebenfalls mit `<depot-id>.ini` gespeichert werden. Hier wird u.a. die Erreichbarkeit des Depots abgelegt.

```
[depotshare]
remoteurl = smb://<NetBiosName>/<Path>
localurl = file://<Path>

[depotserver]
notes = <String>
network = <IP>
description = <String>
hardwareaddress = <MAC>
ipaddress = <IP>
inventorynumber = <String>

[repository]
remoteurl = webdavs://<FQDN>:<Port>/<Path>
localurl = file://<Path>
maxbandwith = <Integer> #in Bytes
```

Hier finden sich aber auch die Informationen, welche opsi-Produkte, in welcher Version und mit welchen Property Defaultwerten, auf dem Depot installiert sind.

Product control files in /var/lib/opsi/config/products/

Die product control files enthalten die Metainformationen der Produkte, wie z.B. Name, Properties und deren Defaultwerte, Abhängigkeiten ...

Die control files entsprechen den control files, wie sie bei der Erstellung von opsi-Produkten im Verzeichnis `<produktname>/OPSI/control` erzeugt werden.

Die control files bestehen aus folgenden Sektionen:

- Sektion [Package]
  Beschreibung der Paketversion und ob es sich um ein incrementelles Paket handelt.
- Sektion [Product]
  Beschreibung des Produktes.
- Sektion(en) [ProductProperty]
  (optional)
  Beschreibung von veränderbaren Produkteigenschaften.
- Sektion(en) [ProductDependency]
  (optional)
  Beschreibung von Produktabhängigkeiten.

Ein Beispiel:

```
[Package]
version: 1
depends:
incremental: False

[Product]
type: localboot
id: thunderbird
name: Mozilla Thunderbird
description: Mailclient von Mozilla.org
advice:
version: 2.0.0.4
priority: 0
```

```
licenseRequired: False
productClasses: Mailclient
setupScript: thunderbird.ins
uninstallScript:
updateScript:
alwaysScript:
onceScript:

[ProductProperty]
name: enigmail
description: Installiere Verschluesselungs Plugin fuer GnuPG
values: on, off
default: off

[ProductDependency]
action: setup
requiredProduct: mshotfix
requiredStatus: installed
requirementType: before
```

- [Package]-*Version*
  ist die Version des Paketes für die Produktversion. Die dient dazu um Pakete mit gleicher Produktversion aber z. B. korrigiertem *opsi-winst*-Skript zu unterscheiden.
- [Package]-*depends*
  gibt bei einem inkrementellen Paket das Basis Paket an, zu dem es inkrementell ist.
- [Package]-*Incremental*
  gibt an ob es ein inkrementelles Paket ist.
- [Product]-*type*
  gibt die Art des Produktes an localboot/netboot.
- [Product]-*Id*
  ist ein eindeutiger Bezeichner für das Produkt in der Regel unabhängig von der Version.
- [Product]-*name*
  ist der Klartextname des Produkts.
- [Product]-*Description*
  ist eine ergänzende Beschreibung zum Produkt, die z.B. im *opsi-configed* unter *Beschreibung* angezeigt wird.
- [Product]-*Advice*
  ist eine ergänzende Beschreibung (in der Regel) zum Umgang mit dem Produkt, die zu beachten ist und im *opsi-configed* unter *Notiz* angezeigt wird.
- [Product]-*version*
  ist die Version der eingepackten Software.
- [Product]-*Priority*
  beeinflusst zusammen mit den Produktabhängigkeiten die Installationsreihenfolge.
- [Product]-*productClasses*
  wird zur Zeit noch nicht verwendet (und auch nicht angezeigt).
- [ProductProperty]-*type*
  Typ des properties: (unicode/boolean)
- [ProductProperty]-*name*:
  Anzeigename der Eigenschaft.
- [ProductProperty]- *multivalue*
  Kann dieses Property eine Liste von Werten enthalten (True/False)

- [ProductProperty]- *editable*
  Kann dieses Property frei editiert werden (oder kann nur aus einer vorgegebnenen Liste ausgewählt werden). (True/False)
- [ProductProperty]-*description*:
  Beschreibung der Eigenschaft (Tooltip im *opsi-configed*).
- [ProductProperty]-*values* :
  Liste möglicher, erlaubte Werte. Wenn leer, dann ist der Wert frei editierbar.
- [ProductProperty]-*default* :
  Default Wert der Eigenschaft.
- [ProductDependency]-*Action* :
  für welche Aktion des Produktes, welches Sie gerade erstellen, soll die Abhängigkeit gelten (setup, uninstall ...).
- [ProductDependency]-*Requiredproduct*:
  Productid (Bezeichner) des Produkts, zu dem eine Abhängigkeit besteht.
- [ProductDependency]-*Required action*:
  Sie können entweder eine Aktion anfordern oder (siehe unten) einen Status. Aktionen können z.B. sein : setup, uninstall, update ...
- [ProductDependency]-*Required installation status*:
  Status, den das Produkt zu dem eine Abhängigkeit besteht, haben soll. Typischerweise *installed* - liegt ein anderer Status vor, so wird das Produkt auf setup gestellt.
- [ProductDependency]-*Requirement type*:
  Installationsreihenfolge. Wenn das Produkt zu dem eine Abhängigkeit besteht installiert sein muss, bevor mit der Installation des aktuellen Produkts begonnen werden kann, dann ist dies *before*. Muss es nach dem aktuellen Produkt installiert werden, so ist dies *after*. Ist die Reihenfolge egal, so muss hier nichts eingetragen werden.

23.4.5 Inventarisierungsdateien /var/lib/opsi/audit

Hier liegen die Dateien der Hardwareinventarisierung (*.hw) und der Softwareinventarisierung (*.sw).

23.5 Dateien des LDAP-Backends

Das opsi-LDAP Schema finden Sie unter `/etc/ldap/schema/opsi.schema`.

23.6 opsi Programme und Libraries

23.6.1 Python Bibliothek

Die opsi Python Module finden sich unter:

- Debian Lenny
  `/usr/share/python-support/python-opsi`
  `/usr/share/python-support/opsiconfd`
- Ubuntu Lucid
  `/usr/share/pyshared/python-opsi`
  `/usr/share/pyshared/opsiconfd`

23.6.2 Programme in /usr/bin

- opsipxeconfd
  opsi Daemon, welcher für den PXE-Start der Clients die notwendigen Dateien im tftp-Bereich des Servers verwaltet.
- opsi-admin
  Kommandozeilen-Interface zur opsi python Library.
- opsiconfd
  opsi Daemon zur Bereitstellung der opsi Methoden als Webservice und vieles mehr.
- opsiconfd-guard
  opsi Daemon, der überwacht, ob der *opsiconfd* läuft und diesen im Zweifelsfall neu startet.
- opsi-configed
  Aufruf des opsi-Managementinterface.
- opsi-convert
  Skript zum Konvertieren zwischen verschiedenen Backends.
- opsi-makeproductfile
  Skript zum opsi-Paket packen.
- opsi-newprod
  Skript zum Erstellen eines neuen Produktes.
- opsi-package-manager
  Skript zum Installieren und Deinstallieren von opsi-Paketen auf einem opsi-server.
- opsi-setup
  Programm für diverse Basiskonfigurationen.

23.7 opsi-Logdateien

Die opsi Logdateien haben das Format:

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

23.7.1 /var/log/opsi/bootimage

Hier findet sich die Logdateien der bootimages zu den Clients. Dabei werden die Dateien als `<IP-Name>.log` angelegt.

Sollte das bootimage den Webservice nicht erreichen können, so findet sich die Logdatei im bootimage unter `/tmp/log`. Um in einem solchen Fall an die Logdatei vom bootimage zu kommen, gibt es zwei Wege:

1. Netzwerk geht
   Dann kann man per `SCP` z.B. von Windows aus per `WinSCP` die Datei `/tmp/log` holen.
2. Netzwerk geht nicht

Dann hilft der USB-Stick:

- Als root mit pass *linux123* einloggen
- USB-Stick einstecken und ein paar Sekunden warten
- mit `sfdisk -l` prüfen, auf welchem Device der Stick liegt
- mounten
- kopieren
- unmounten

Das Ganze sieht als Beispiel etwa so aus:

```
#sfdisk -l
Disk /dev/sda: 30401 cylinders, 255 heads, 63 sectors/track
Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting from 0

   Device Boot Start     End   #cyls    #blocks   Id  System
/dev/sda1   *      0+  30401-  30402- 244197528+   7  HPFS/NTFS
/dev/sda2          0       -       0          0    0  Empty
/dev/sda3          0       -       0          0    0  Empty
/dev/sda4          0       -       0          0    0  Empty

Disk /dev/sdb: 1017 cylinders, 33 heads, 61 sectors/track
Units = cylinders of 1030656 bytes, blocks of 1024 bytes, counting from 0

   Device Boot Start     End   #cyls    #blocks   Id  System
/dev/sdb1          0+   1016    1017-   1023580    b  W95 FAT32
/dev/sdb2          0       -       0          0    0  Empty
/dev/sdb3          0       -       0          0    0  Empty
/dev/sdb4          0       -       0          0    0  Empty
# mount /dev/sdb1 /mnt
# cp /tmp/log /mnt
#umount /mnt
```

23.7.2 /var/log/opsi/clientconnect

Hier findet sich die Logdatei der auf dem Client laufenden *opsi-client-agent*.
Dies ist auf dem client die `C:\tmp\opsiclientd.log`.

23.7.3 /var/log/opsi/instlog

Hier findet sich die Logdatei der auf den Clients ausgeführten *opsi-winst*-Skripte.
Die Originale liegen auf dem Client unter `c:\tmp\instlog.txt`

23.7.4 /var/log/opsi/opsiconfd

Hier findet sich die Logdatei des *opsiconfd* selbst sowie log Dateien zu den Clients.
Dabei werden die Dateien als `<IP-Nummer>.log` angelegt und soweit in `/etc/opsi/opsiconfd.conf` eingestellt, zu diesen symbolische Links als `<IP-Name>.log` erzeugt.

23.7.5 /var/log/opsi/opsipxeconfd.log

Logdatei des *opsipxeconfd*
welcher für den PXE-Start der Clients die notwendiogen Dateien im tftp-Bereich des Servers verwaltet.

23.7.6 /var/log/opsi/package.log

Logdatei des opsi-package-manager.

23.7.7 /var/log/opsi/opsi-product-updater.log

Logdatei des opsi-product-updater.

23.7.8 tftp log in /var/log/syslog

Die Logeinträge des tftpd finden sich in `/var/log/syslog`.

Damit diese auch aussagerkräftig sind muss der Loglevel des tftpd auf 7 erhöht werden:
In der Datei `/etc/inetd.conf` in der Zeile die mit *tftpd* anfängt den Parameter *verbose* auf 7 setzen:

```
tftp    dgram   udp     wait    nobody /usr/sbin/tcpd /usr/sbin/in.tftpd --tftpd-timeout 300 --retry-timeout 5     --\
    mcast-port 1758 --mcast-addr 239.239.239.0-255 --mcast-ttl 1 --maxthread 100 --verbose=7  /tftpboot
```

danach ausführen:

```
killall tftpd
killall -1 inetd
```

23.7.9 c:\tmp\opsiloginblocker.txt

Logdatei des Loginblockers.

23.7.10 c:\tmp\opsiclientd.log

Logdatei des opsiclientd.
Wird bei Beendigung auf den Server nach `/var/log/opsi/clientconnect/<pc-ipnummer.log>` kopiert.

23.7.11 c:\tmp\instlog.txt

Logdatei des *opsi-winst*.
Wird bei Beendigung auf den Server nach `/var/log/opsi/instlog/<pc-ipnummer.log>` kopiert.

24 Registryeinträge

24.1 Registryeinträge des opsiclientd

24.1.1 opsi.org/general

- `bootmode= <bkstd | reins>`
  Beschreibt ob der Rechner gerade aus einer Reinstallation kommt.

24.1.2 opsi.org/opsi-client-agent und opsi.org/preloginloader

Schlüssel [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\opsi-client-agent] Schlüssel [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\preloginloader]

Diese beiden Keys sind identisch, wobei der zweite (preloginloader) abgekündigt ist und nur der Rückwärtskompatibilität dient.

- `"RemoveMsginaOnDeinst"=dword:00000001`
  - bei Reinstallation msgina deinstallieren
- `"WinstRegKey"="SOFTWARE\\opsi.org\\winst"`
  - hier wird geschaut, ob der *opsi-winst* rebooten möchte
- `"LoginBlockerStart"=dword:00000001`
  - opsigina wartet auf READY aus der Named Pipe
- `"LoginBlockerTimeoutConnect"=dword:00000005`
  - Timeout in Minuten für ein Pipe-Connect zum opsi-client-agent
  - nur wenn LoginBlockerStart auf 1 steht
- `"LoginBlockerLogLevel"=reg_dword:006`
  - Loglevel des Loginblockers
- `"OpsiServiceType"=dword:00000002`
  - Soll sich die opsigina mit dem prelogin (1) oder opsiclientd (2) unterhalten
- `"NextGina"="msgina.dll"`
  - die nächste gechainte gina.dll

24.1.3 opsi.org/shareinfo

- `depoturl`
  <Url die zu den Softwarepaketen verweist. Muster: protokoll:\\server\share\dir>
  Beispiel:
  smb:\\schleppi\opsi_depot
- `depotdrive`
  <Laufwerksbuchstaben auf den depoturl gemountet wird>
  Beispiel: `P:` (mit Doppelpunkt)

24.2 Registryeinträge des opsi-winst

24.2.1 opsi.org/winst

Diese Registry Einträge werden vom Winst selbst verwaltet und sollten nicht verändert werden.

```
"LastLogFilename"="C:\\TMP\\syslogin.log"
"ContinueLogFile"=dword:00000000
"RebootRequested"=dword:00000000
"SendLogToService"=dword:00000001
"NumberOfErrors"=dword:00000000
"ShutdownRequested"=dword:00000000
```

24.2.2 Steuerung des Logging per syslog-Protokoll

Schlüssel HKLM\Software\opsi.org\syslogd

DWord-Variable remoteerrorlogging wird nach folgendem Schema ausgewertet:
RemoteErrorLogging = (trel_none, trel_filesystem, trel_syslog);

Falls das Logging mittels dem syslog-Protokoll erfolgt ("remoteerrorlogging"=dword:00000002), so gibt die String-Variable sysloghost den IP-Namen des LogHost an. DWORD-Variable syslogfacility gibt an, was als Quelle der syslog-Nachricht übergeben wird. Der default ist ID_SYSLOG_FACILITY_LOCAL0 Bedeutungen:

```
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)
```

25 Upgrade Anleitungen für den opsi-server

Diese finden Sie ab opsi 4.0 in den versionspezifischen *releasenotes-upgrade* Handbüchern.