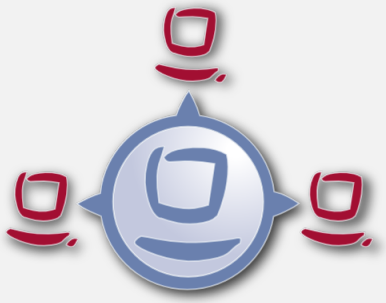


opsi manual opsi version 4.0.1

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

Stand: 05.05.2011

Inhoudsopgave

1 Introduction

1.1 Who should read this manual?

This manual is written for all who want to gain a deeper insight into the mechanisms and the tools of the client management system opsi (öpen pc server integration").

It presents a complete HOWTO for the use of opsi while emphasizing the understanding of the technical background. The decision maker who decides on using opsi as well as the system administrator who works with it will get a solid foundation for their tasks.

1.2 Notations

Angle brackets < > mark abstract names. In a concrete context any marked *<abstract name>* must be replaced by some real name. Example: The file share, where opsi places the software packets, may abstractly be noted as *<opsi-depot-share>*. If the real fileshare is `/opt/pcbin/install`, then you have to replace the abstract name by exactly this string. The location of the packet `<opsi-depot-share>/ooffice` becomes `/opt/pcbin/install/ooffice`.

Example snippets from program code or configuration files use a Courier font, with a background color:

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

2 Overview of opsi

Tools for automated software distribution and operating system installation are important and necessary tools for standardization, maintainability and cost saving of larger PC networks. Normally the application of such tools comes along with substantial royalties, whereas opsi as an open source tool affords explicit economics. Expenses thereby arise only from performed services like consulting, training and maintenance, and perhaps from low Co-funding rates if you like to use some of the non free modules.

Although the software itself and the handbooks are free of charge, the process of introducing any software distribution tool is still an investment. To get the benefit without throwbacks and without a long learning curve consulting and education of the system administrators by a professional partner is recommended. uib offers all these services around opsi.

The opsi system as developed by uib depends on Linux-servers. They are used for remote installation and maintenance of the client OS and the client software packets ("PC-Server-Integration"). It is based as far as possible on free available tools (GNUtools, SAMBA etc.). The complete system all together is named opsi (Open PC-Server-Integration) and with its configurability is a very interesting solution for the administration challenges of a large computer park.

2.1 Experience

opsi is derived from a system, which is in use since the middle of the 90's with more than 2000 Client-PCs in different locations of a state authority. Since that time it has continuously been adapted to the changing Microsoft operating system world. As a product opsi is now accessible for a broad range of interested users.

You can find an geographical overview of the registered opsi-installations at: http://www.opsi.org/map/.

2.2 opsi features

The core features of opsi are:

- automatic software distribution
- automatic operating system installation

- hard- and software inventory with history
- comfortable control via the opsi management interface
- support of multiple depot-servers

2.3 opsi Extensions

- Management of licenses
- MySQL-Backend
- Use of hierarchical client groups (Treeview)
- Dynamical depot server selection
- Software on Demand
- Support for clients behind slow connections (WAN Extension)

3 opsi configuration and tools

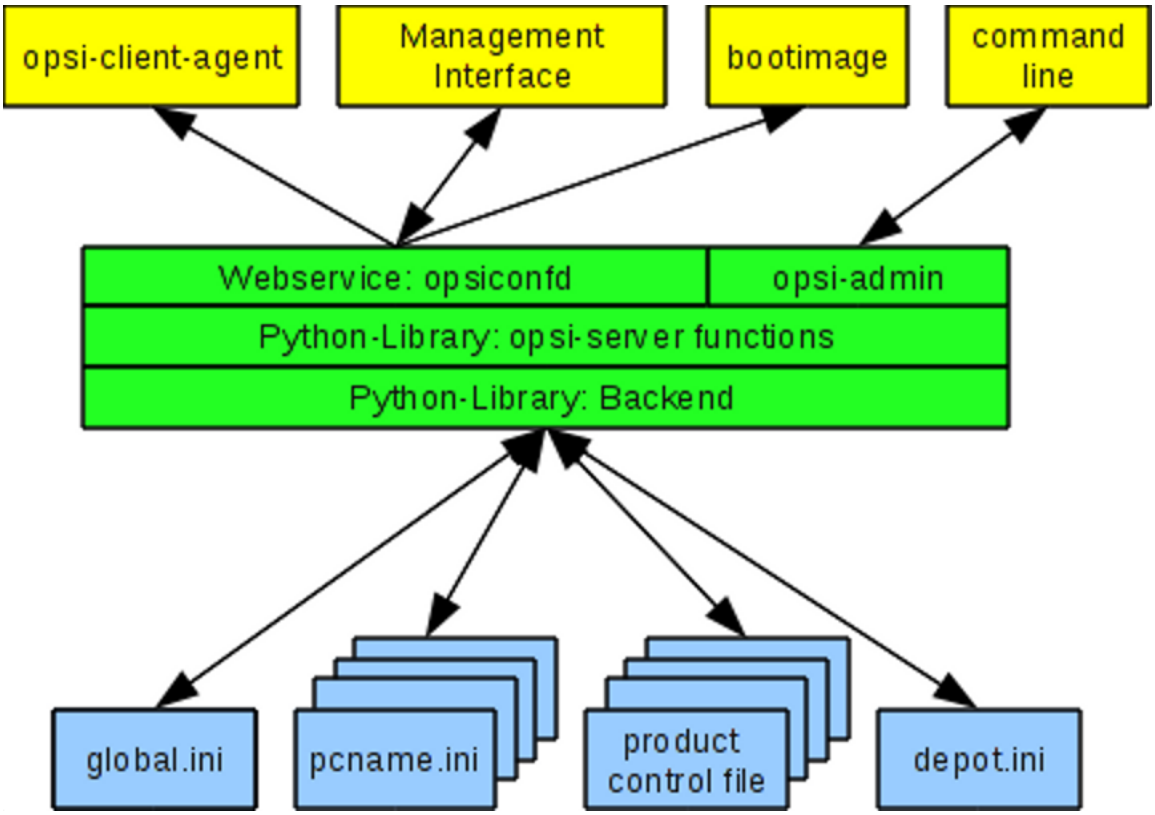
3.1 Overview

The configuration of opsi requires some data management. All non-server components are using a web service for data exchange with the opsi server. They exchange data via the *opsiconfd*, and the *opsiconfd* forwards the data to the backend manager which passes the data into the selected backend.

opsi supports different backends: Backends:

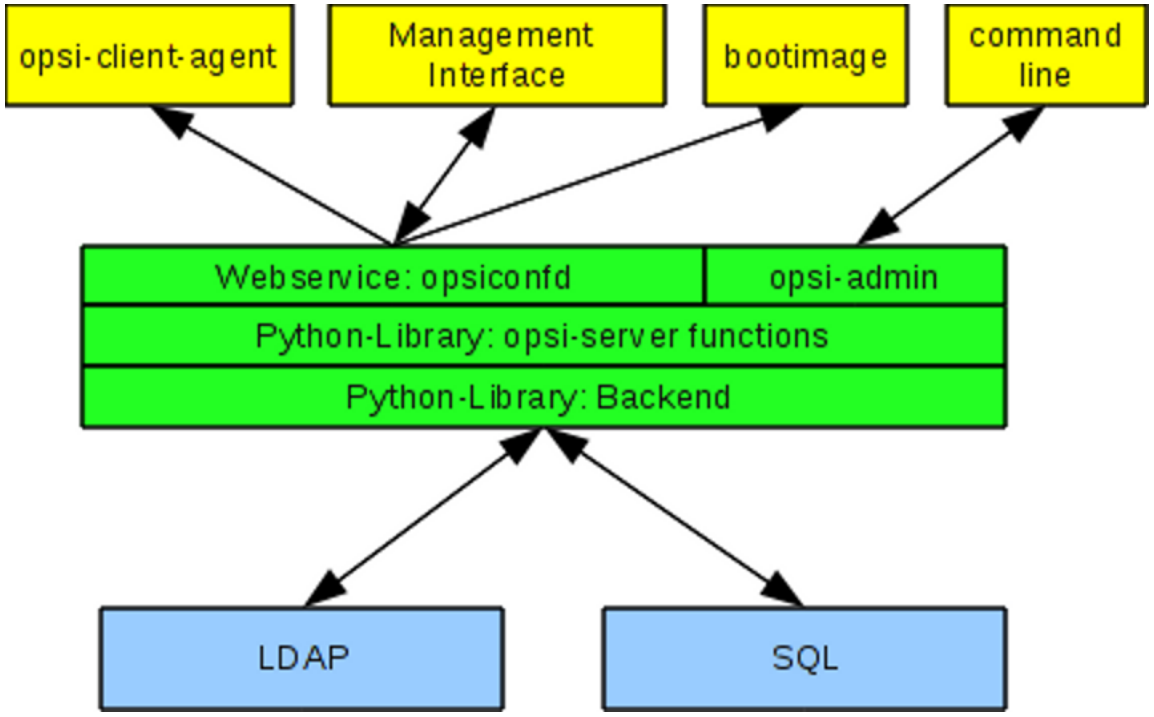
- File based
- LDAP based
- MySQL based

Using the file backend the data are stored in ini like text files.



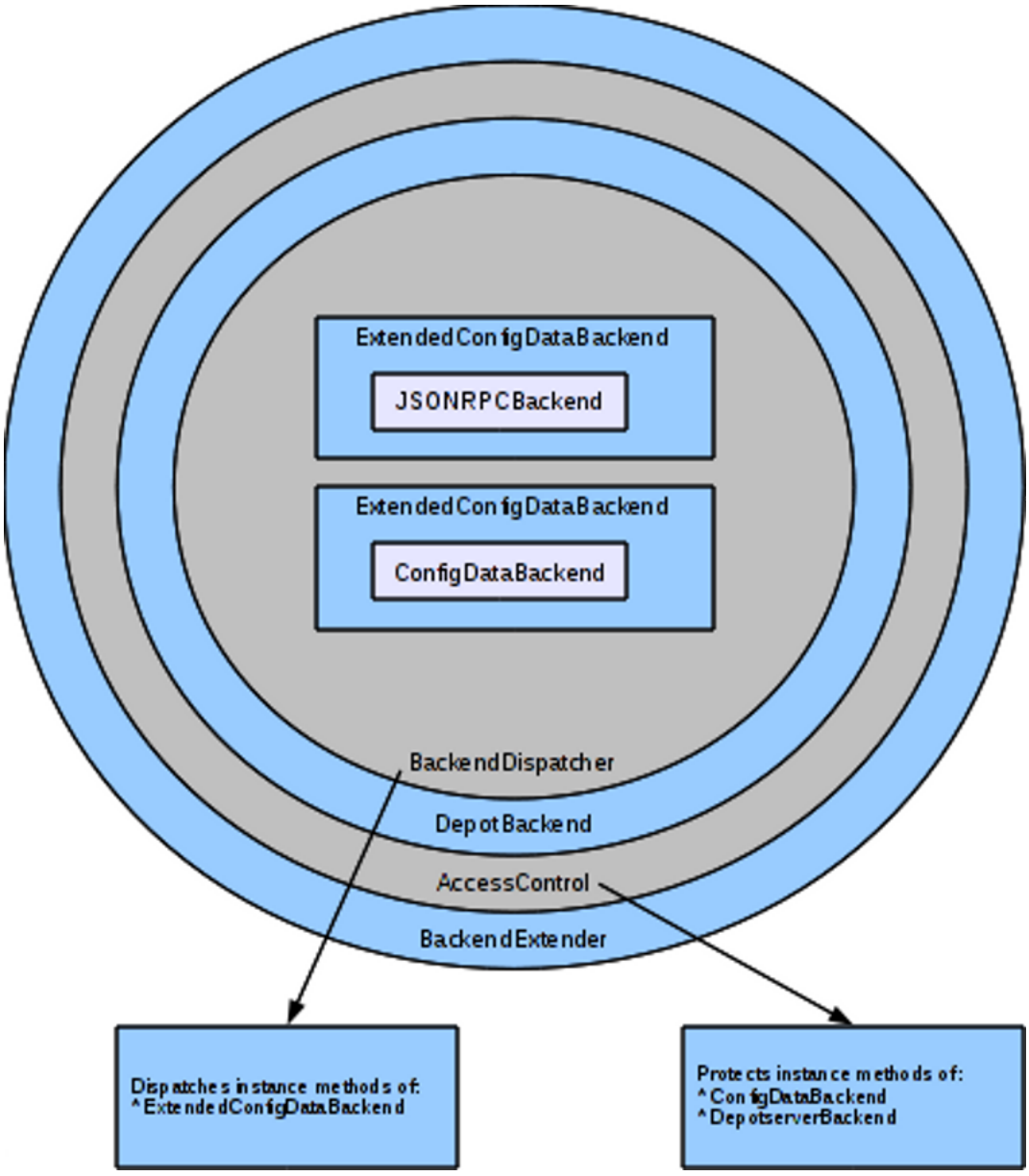
Figuur 1: Scheme: opsi with file backend

Using the mysql or ldap backend the data are stored in specific data objects.



Figuur 2: Scheme: opsi with SQL / LDAP backend

More details you will find at



Figuur 3: Scheme: backend layers and access control

The in opsi 3 used directory `/etc/opsi/backendManager.d` isn't used in opsi 4 anymore.

The configuration files in `/etc/opsi/backends` define the backends.

Which backend is used for which data, is configured in the file `/etc/opsi/backendManager/dispatch.conf`.

The file `/etc/opsi/backendManager/acl.conf` defines who has access to which methods.

Below the directory `/etc/opsi/backendManager/extend.d` there could be files which defines extended opsi methods. So you will find here for example the files which define the old opsi 3 *legacy* methods by mapping them to the new opsi 4 methods (`/etc/opsi/backendManager/extend.d/20_legacy.conf`).

A more detailed reference of these configuration files you will find at

3.2 Tool: opsi-setup

This program is something like the *swiss army knife* of the opsi configuration. It is used by the opsi installation scripts and can be also called separately for maintanace and repair purpose.

The tasks of opsi-setup are:

- register a opsi-server as depot server
- correct file access rights
- initialize data storage backends
- upgrade backend (from 3.4 to 4.0)
- setup of the MySQL-backend
- edit the default configurations
- cleanup the current backend(s)
- configure the essential samba shares
- configure the essential dhcp entries

The command `opsi-setup --help` shows the program options:

```
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
   --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
```

The functions and options in detail:

- `--register-depot`
  This option is used to register a *opsi-server* as depot server to a other *opsi-server* (*opsi-configserver*). For details see

- `--set-rights [path]`
  Sets the file access rights in all opsi directories:
  - `/tftpboot/linux`
  - `/home/opsiproducts`
  - `/var/log/opsi`
  - `/var/lib/opsi`
  - `/opt/pcbin/install`

  - `/etc/opsi`
    You may give a directory name as argument to set only the access rights below this directory.
    e.g.
    `opsi-setup --set-rights /opt/pcbin/install/winxppro/drivers`

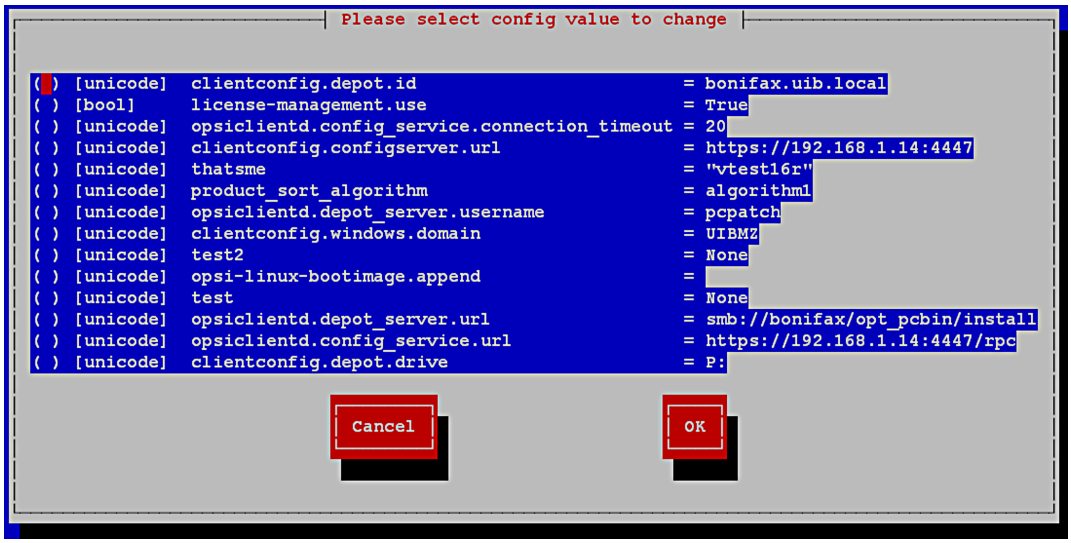
- `--init-current-config`
  initialize the configured backend. Should be always called after changing the file
  `/etc/opsi/backendManager/dispatch.conf`

- The three commands:
  `--update-mysql`
  `--update-ldap`
  `--update-file`
  are used to upgrade the backends from one opsi release to the next one.
  For details see the *releasenotes-upgrade-manual*.

- `--configure-mysql`
  does the first time database setup.

- `--edit-config-defaults`
  To edit the default values of some configuration data like in the *server configuration* of the opsi-configed.

- `--edit-config-defaults`
  To edit the default values of some configuration data like in the *server configuration* of the *opsi-configed*.



Figuur 4: Dialog: `opsi-setup --edit-config-defaults`

e.g.:

clientconfig.depot.id
The name of the default depot server.

license-management.use
Defines if netboot products should get license keys from license management or from product properties.

product_sort_algorithm
Defines the algorithm which is used to calculate the product installation sequence.

- `--cleanup-backend`
  Check the current backend(s) for entries which are not needed anymore and referential integrity

- `--auto-configure-samba`
  Creates the opsi share entries in the `/etc/samba/smb.conf` configuration file
- `--auto-configure-dhcpd`
  Creates the by opsi needed entries in the `/etc/dhcp3/dhcpd.conf.
  Don't use this if you not plan to use the dhcpd on the opsi server.
  More details in the *opsi-getting-started* manual

3.3 Tool: Management Interface: *opsi-configed*

3.3.1 Requirements and operation

The *opsi-configed* requires Java 1.6 and a running `opsiconfd` on the server.

If you are running the *opsi-configed* on a Linux based machine, so make sure that your Java is the *Sun* Java Version. The often installed OpenJDK or other versions may lead to subtil errors. So you have to install the Sun Java and configure it as the default Java:

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

The command

```
java -version
```

should lead to the following output:

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

Most times the opsi-configed will be called as applet in the browser via: `https://<servername>:4447/configed`

The *opsi-configed* as application is also part of the opsi product *opsi-adminutils* and may then be started via the windows start menue. At the server the *opsi-configed* is installed as part of the opsi-server installation. It may be started using the menue entry or with the command `/usr/bin/opsi-configed`.

If you in the correct directory, it also can be started with `java -jar configed.jar`.

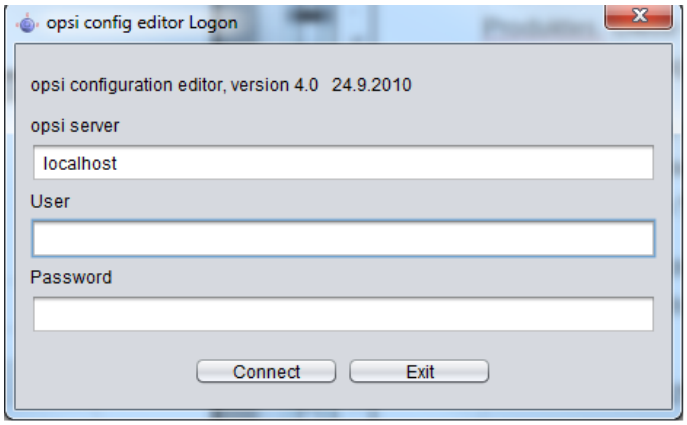
The help option `java -jar configed.jar --help` shows the available command line options.

```
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
```

3.3.2 Login



Figuur 5: *opsi-configed*: login mask

At login time the opsi-configed tries to connect the opsi server via https. The login is done with the given parameters *opsi server[:Port]* (default port 4447 – opsiconfd) and the User/Password of the *opsi-configserver* account. For a successful login the provided user has to be a member of the unix-group *opsiadmin*.

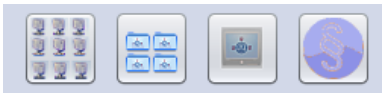
3.3.3 Copy & Paste, Drag & Drop

You may copy the selected entries from nearly every section of the *opsi-configed* to the clipboard using the standard key combinations (*Strg-Insert*, *Strg-C*). This may be used to transfer interesting data to other programs. For the most tables you may also use *Drag & Drop* to copy the data to programs like *Excel*.

Opmerking
Since Java version 1.6.24 Oracle has deactivated the *Copy & Paste* from a Java Applet in the browser for security reasons. But *Drag & Drop* is still working'

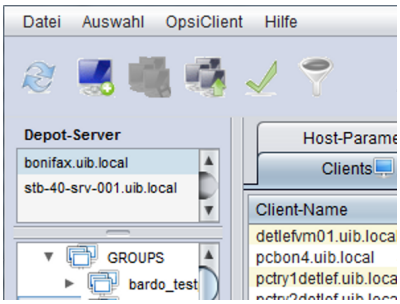
3.3.4 Client configuration / server configuration / license management

To switch between the different views of the *opsi-configed*, use the buttons in the upper right corner.



Figuur 6: opsi-configed: Buttons for (from left to right): Client configuration, Server configuration, License management

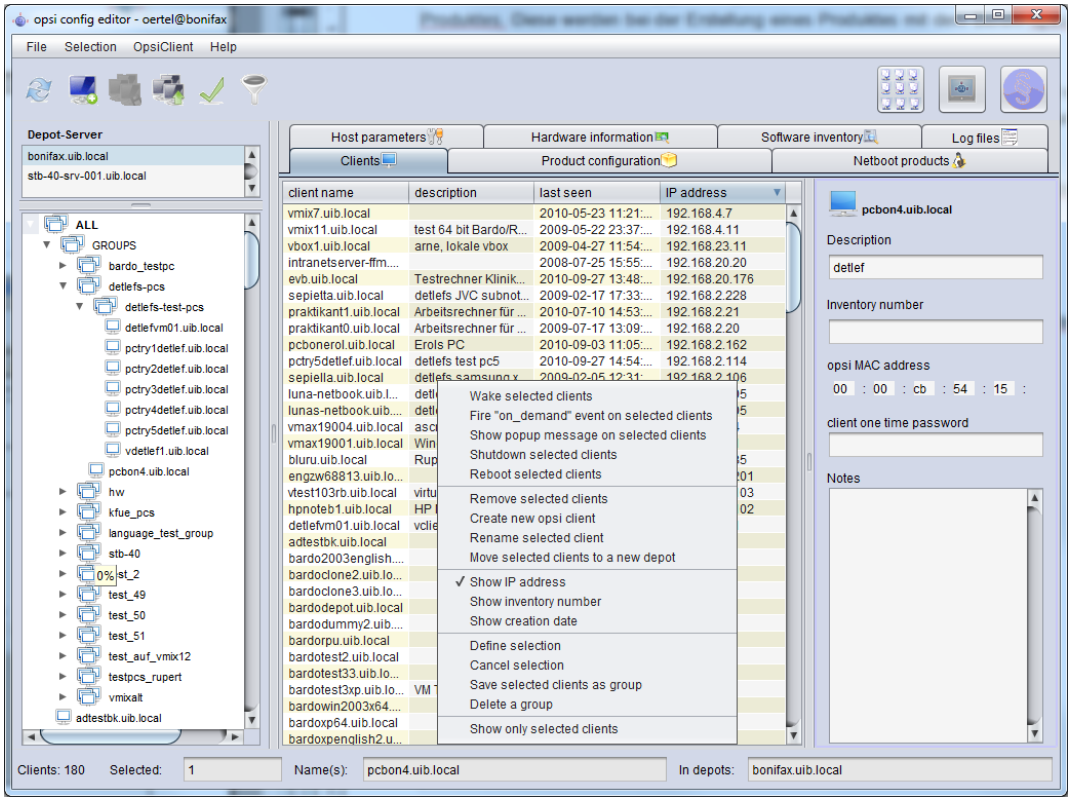
3.3.5 Depot selection



Figuur 7: *opsi-configed*: depot selection

3.3.6 Single client selection and group configuration

After a successful login the main window pops up and shows the tab *Client selection*. This tab shows a list of known clients from the selected *opsi-depot* or the clients which are selected using the *treeview control* on the left side of the *opsi-configed*.



Figuur 8: *opsi-configed*: client selection mask

The clients list

The clients list has per default the columns *client name*, *description*, *On* and *last seen*.

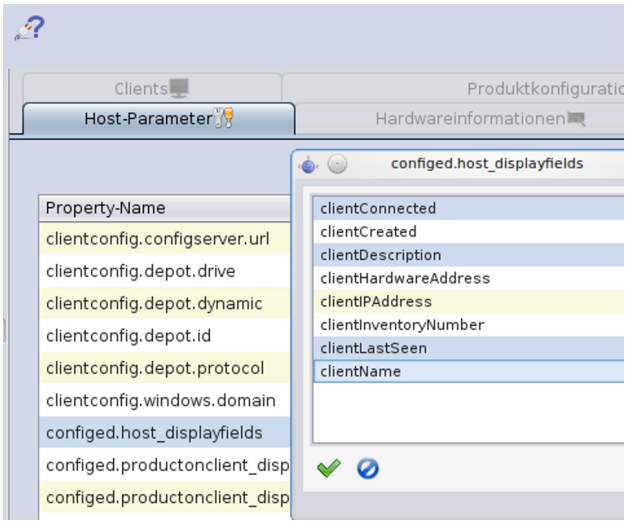
- *client name* is the *full qualified hostname* which is the client name including the domain name

- *description* is a free selectable description which you can edit in the right top part of the window
- *On* shows after clicking the button *Check wich clients are connected* the result of this query.
- *last seen* shows the date and a time of the last client connect to the opsiconfd web service

Some columns are deactivated by default: * *IP address* shows the IP-Number. * *Inventory No* shows the (optional) given Inventory Number. * *created* shows the date and a time of the client creation. It isn't visible by default and have to be activated by the context menu. * *opsi mac address* shows the MAC of the client

You may activate these columns using the context menu. The configuration which columns are activated may be changed using the *host parameter configed.host_displayfields*.

Figuur 9: *opsi-configed*: Button *Check wich clients are connected*



Figuur 10: *opsi-configed*: change the default for visible columns in the clients list

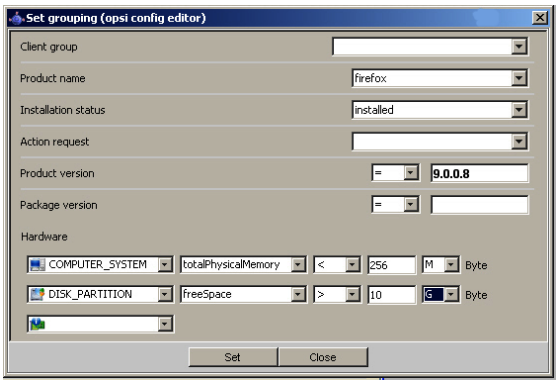
To sort the clients by a certain column click on the top header of that column.

Selecting clients

You can select one or multiple clients to work with. The client view can be restricted to the selected clients by clicking the funnel icon or from the menu by *Grouping / Show only selected clients*.

A selected client group can be saved with the icon *Save grouping* or from the menu by *Grouping / save group* with a free selectable name.

With the icon *Set client group* or *Grouping / set client group* saved groups can be loaded.



Figuur 11: *opsi-configed*: mask: group setting

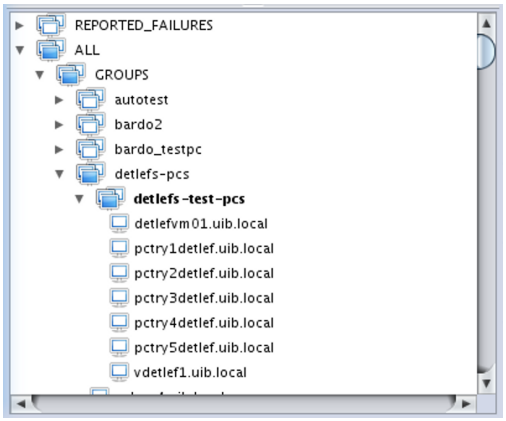
With the function *Set client group* you can build client groups by certain criteria (e.g.: all clients which have the product *firefox* with the installation status *installed*).

3.3.7 Client selection and hierarchical groups using the treeview

Since opsi 4.0 it is possible to manage groups and clients using a tree view control on the left side of the *opsi-configed*. A second enhancement is the possibility of hierarchical groups (groups in groups). This tree view feature is part of a co-funding project and runs only with a valid activation file. A activation costs 500 €. For evaluation please contact info@uib.de. The tree view control has base node *ALL* with all groups and clients beyond..

Basic concepts

The tree view control has base node *ALL* with all groups and clients beyond. Ther is a other node *Groups* which is the bas group for all other self defined groups.



Figuur 12: *opsi-configed*: Treeview with clients and groups

There is a additional group *REPORTED_FAILURES* which contains all clients, which have a action result *failed*.

Every known client is alwas in the group *ALL*. Add itionally a client may be in one or more other groups. You may build up different group trees which represent different order critiras like administrative structure, hardware or typical software inventory.

If you select a client, all groups where the selected client belog to get colored marked icons.

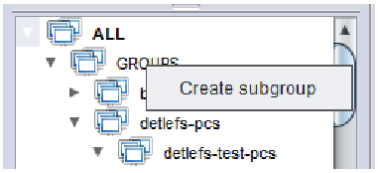
How to ...

By a click one a node (or a group) all clients beyond this node will be shown in the *Clients* tab, but none of these clients is selected for processing.

By a click one a client, this client will be shown in the *Clients* tab and selected for processing. You may also use this way to change the selected client while you are in a other tab like product configuration without coming back to the clients tab.

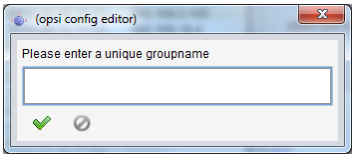
You may use Ctrl-click and Shift-click to select multiple clients. This tree view control show the groups which are created according the chapter

You may also create groups by using the context menu above *ALL* or any existing group.



Figuur 13: *opsi-configed*: Using the context menu to create a new subgroup

You will be asked for the new groups name.



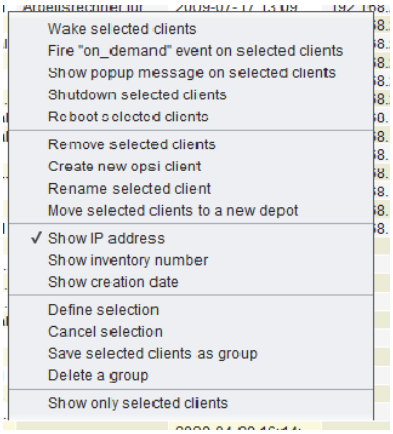
Figuur 14: *opsi-configed*: Dialog: Group name

A group can be populated with clients using Drag&Drop by

- copying clients from the *Clients* tab to the group in the tree view (left mouse button)
- copying clients from the tree view control below the node *ALL* to group in the tree view (left mouse button)
- moving clients from a group in the tree view control to a other group in the tree view (left mouse button)
- copying clients from a group in the tree view control to a other group in the tree view (Ctrl-left mouse button)

3.3.8 Client processing / Client actions

Using the menu *OpsiClient* or the context menu in the *Clients* tab you may choose from a lot of client specific operations



Figuur 15: *opsi-configed*: : context menu *Clients* Tab

WakeOnLan (*Wake selected clients*)

Choosing this menu entry, you will send the selected clients a WakeOnLan signal.

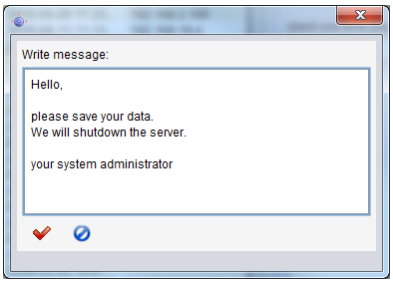
Fire *on_demand* event (Push Installation)

This menu entry is used to send to the *opsi-client-agent* on the selected clients a command to fire the event *on_demand*. This event will start the processing of the current set action request immediately. All messages will be shown on the active desktop. If the client isn't reachable, you will get a message.

What happens exactly if you fire the event *on_demand* can be configured in the event *on_demand* configuration.

Sending messages (*Show popup message*)

Choosing the menu entry *Show popup message* you will get a small edit window where you can type in your message.



Figuur 16: *opsi-configed*: opsi message edit mask

By clicking on the red tick you will send the message to the selected clients.

At the selected clients a message window will appear.

Figuur 17: *opsi-configed*: opsi message display dialog

Shutdown / reboot of selected clients

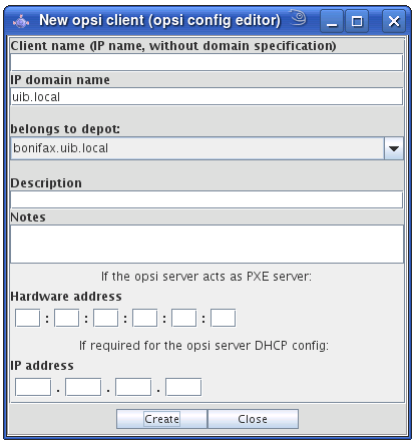
You may sen the selected clients a shutdown or reboot signal. You have to confirm this command at the opsi-configed.

Let op
If the client received the signal, it will going down with out any more questions.

Delete, create, rename and move clients

You may delete the selected clients from the *opsi-server*.

If you choose to create a client, an input mask opens. There you enter or confirm the required data – client name without domain specification, domain name, depot server name. You may add a textual description for this client and notes on this client.

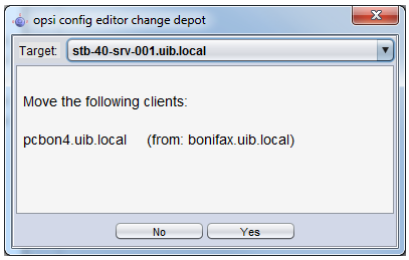


Figuur 18: *opsi-configed*: creating a client

The mask also contains fields for an optional declaration of the IP-number and the ethernet (MAC) address of a client. If the backend is activated for the configuration of a local dhcp-server (which is not the default setting), this information will be used to make the new client known to the dhcp-server. Otherwise the MAC address will be saved in the backend and the IP-number will be discarded.

You may rename a selected client, you will be asked for the new name.

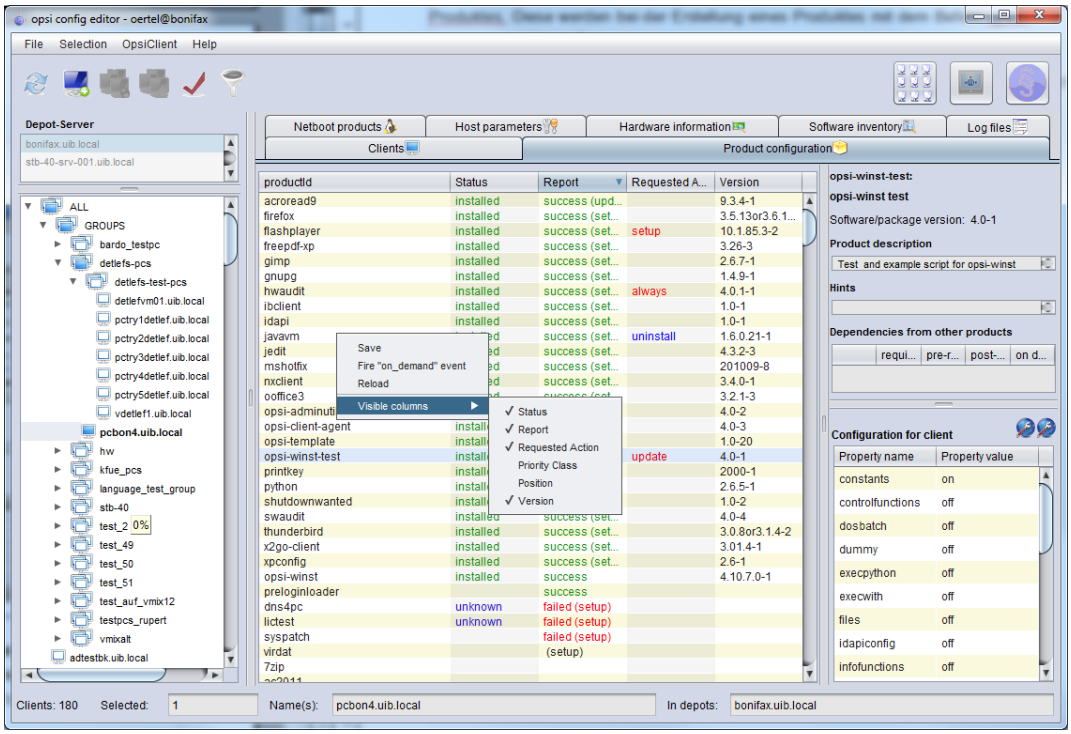
Moving a client to a different depot-server. If clicked the following windows appears with a list of existing depot-servers



Figuur 19: *opsi-configed*: change the depot of a client

3.3.9 Product configuration

Switching to the tab *Product configuration* you get a list of available software packets with its installation status and action status for the selected clients.



Figuur 20: *opsi-configed*: product configuration mask

If there is a different status for the selected clients this will be marked grey (*undefined*). The list of the selected clients is shown at right on top.

You can also sort the product list by clicking at the column header.

This are the columns:

- *Status* is the last announced state of the product and can hold the values *installed*, *not_installed*, *unknown*. The table shows an empty cell if the value is *not_installed* to improve the usability of the view. The cell becomes grey if a multitude of selected clients is selected and does not share a common value (grey coloring represents the pseudo value *mixed*).
- *Report* informs about the progress or the result of the last action using the pattern <action result> (<last action>). During an installation process there may be indicated *installing*, afterward e. g. *failed(setup)* or *success (uninstall)*.

- The column *Requested action* holds the information which action is to be executed. Possible values are *none* (shown by an empty cell) and the action types for which scripts are defined in the product package (possible values are *setup*, *uninstall*, *update*, *once*, *always*, *custom*).
- The field *Version* displays the software version number combined with the opsi package number of the software package installed on the client.

There are two more columns which can be activated via the context menu:

- *Priority class* displays a priority value that is assigned to the product (highest priority +100, lowest priority -100). It influences the product order when products are installed (by virtue of the product_sort_algorithm)
- The *position* column displays the product ordering forecast for installation sequences.

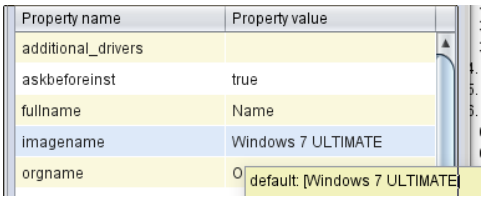
Choose a software product to get more product information in the right part of the window like:

- *Complete product name*: full product name of that software package.
- *Software/package version*: *software version-version of the opsi package* of the software package (specified in the opsi installation package).
- *Product description*: free text to describe the software.
- *Hints*: free text with advices and caveats for handling the package.
- *Requirements*: A list of other products which the selected product (say *A*) depends on combined with the type of dependency: *required* means that A requires the other product (*B*), but it doesn't matter whether B is installed before or after A. *pre-required* means B has to be installed before A. *post-required* means B needs to be installed A. *on deinstall* means this action should take place if A be de-installed.
- *Configuration for client*: It is possible to define additional properties for a product. Their values can be evaluated in a setup script to configure the product per client. Because of the intrinsic complexity of a property definition there is a specific GUI element for displaying and editing the table of properties:

3.3.10 Property tables with list editor windows

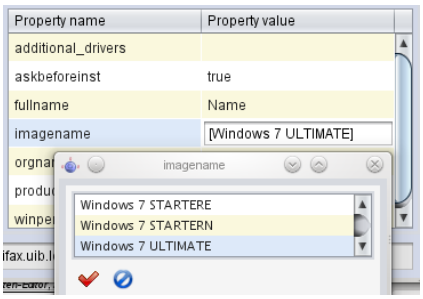
A property table is a two-column table. In each row, the first column contains a property name, the second column displays the assigned property value(s).

It may be configured that a tool tip is displayed showing some information on the meaning of the property and the default value.



Figuur 21: *opsi-configed*: property table with tooltip

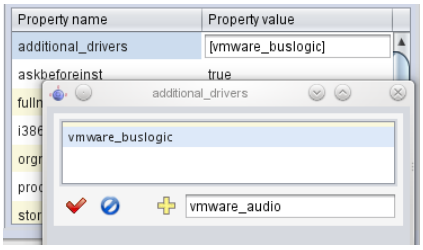
If you click at a value a window pops up: the *list editor* for this property. It shows a value resp. a list of preconfigured values with the current value as selected.



Figuur 22: *opsi-configed*: list editor, selection list

Clicking a new value changes the selection.

If the property value list is editable (new values may be added to the existing list resp. existing values changed) the window comes up with an edit field for the new or modified values.



Figuur 23: *opsi-configed*: list editor, edit field

The most comfortable way to get a new value that is a variant of an existing one is double clicking the existing value in the list. This copies it into the edit field where it can be modified.

As soon as the edit field contains a new value – not yet occuring in the value list – the plus button is activated by which the new value can be added to the list of values.

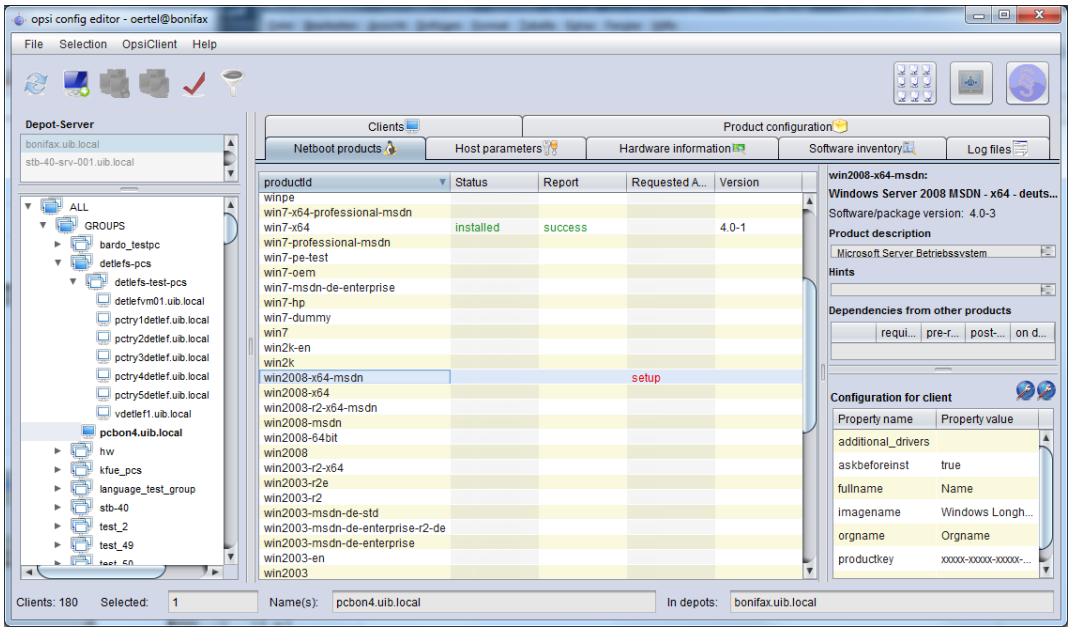
If multiple values are allowed – as it should be e.g. for the property *additional drivers* – a value may be added to the set of selected values by Strg-Click . The very same action removes the value from the set. The minus button (since opsi-configed version 4.0.2) clears the selection completely.

When the list has been edited the green check mark turns to red as usual in the opsi-configed. Clicking it takes the new selection as new property value (and finishes editing). Clicking the blue cancel button stops editing and resets the original value.

3.3.11 Netboot products

The products on tab *Netboot products* are mainly used to install the client OS (operating system) and are listed and configured like the products on tab *Product configuration*.

If for the selected client(s) a netboot product is set to *setup*, the correspondent bootimage will be loaded and executed at the next client reboot.

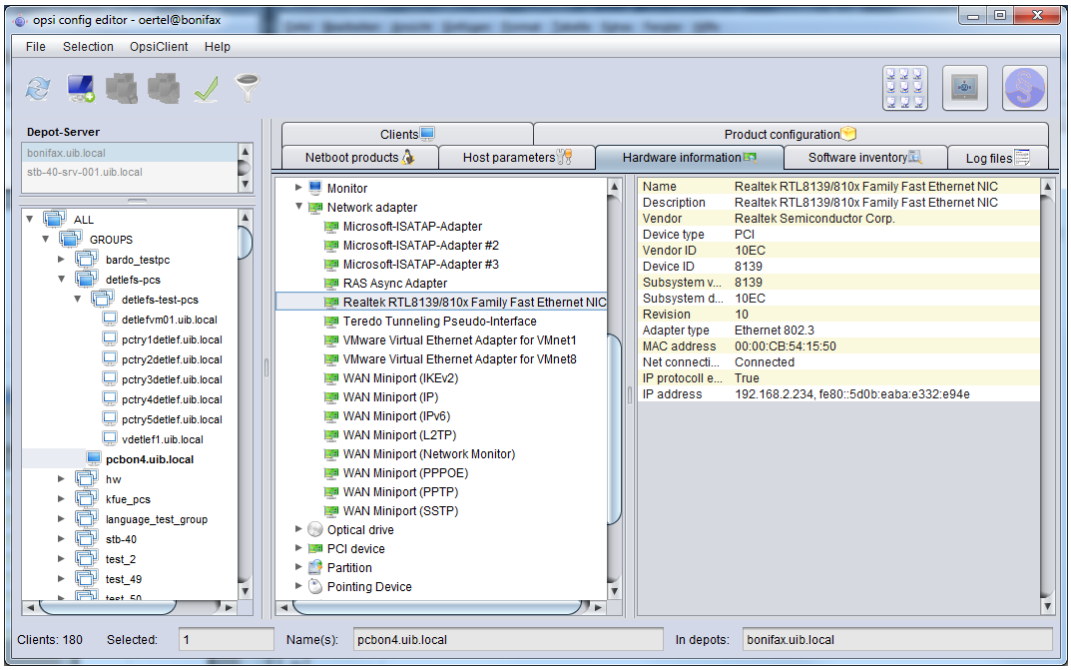


Figuur 24: *opsi-configed*: mask to start the bootimage

This is usually done to initiate an OS installation or any other bootimage task (like a memory test etc.)

3.3.12 Hardware information

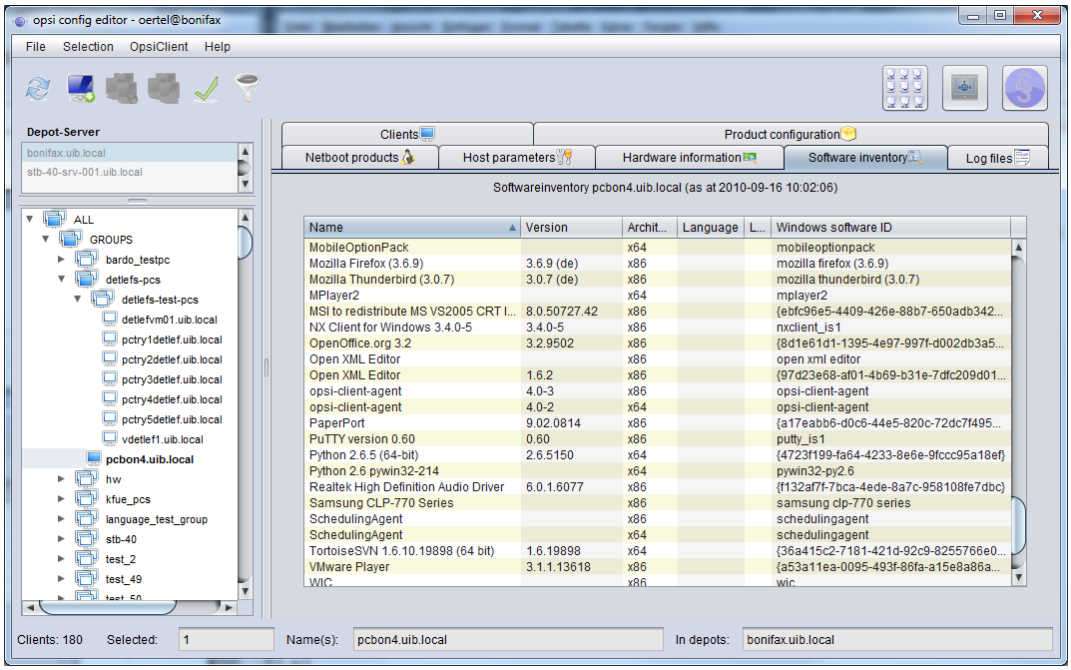
With this tab you get the last detected hardware information for this client (only available if a single client is selected).



Figuur 25: *opsi-configed*: Hardware informations for the selected client

3.3.13 Software inventory

With this tab you get the last known software information for this client (only available if a single client is selected).

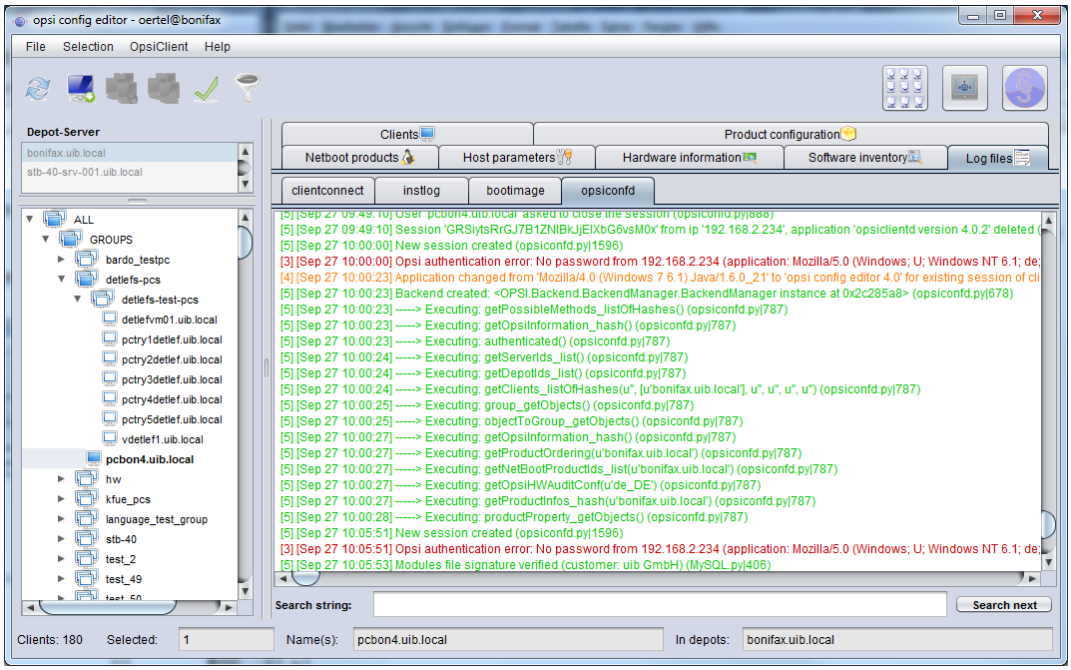


Figuur 26: *opsi-configed*: Software information for the selected client

3.3.14 3.3.13. Logfiles: Logs from client and server

The client specific log files are stored on the server and visible with the opsi-configed via the Tab *log files*.

It's also possible too search in the log file (to continue the search press *F3* or *n*).



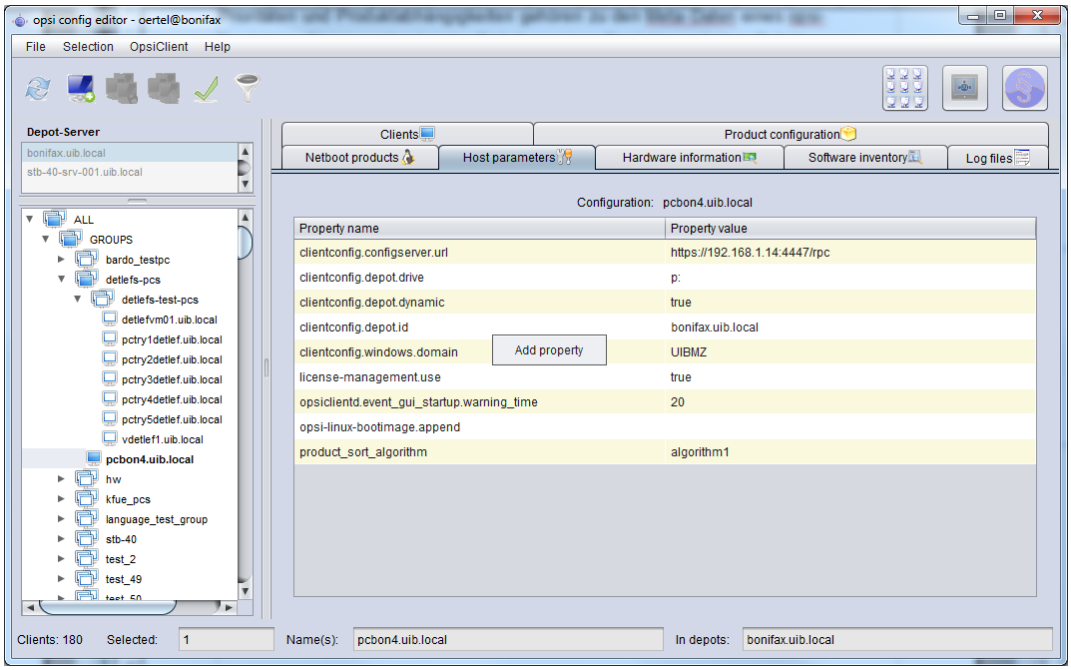
Figuur 27: *opsi-configed*: Display of the log file in the opsi-configed

3.3.15 Host parameter at the client and the server configuration

With the tab *Host parameter* you can provide settings for the general configuration of opsi and other optional configurations. This may be done for the server defaults in the mode *server configuration* or client specific in the mode *client configuration*.

see also:

You may use the context menu to create additional entries.

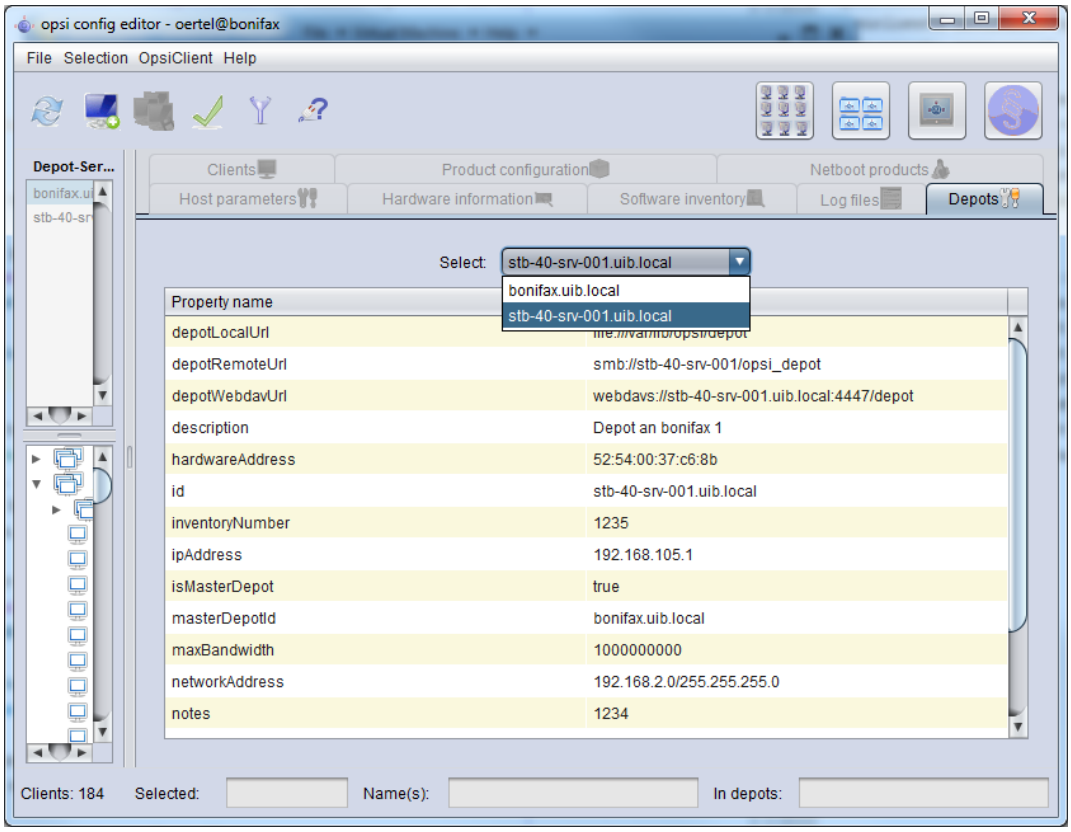


Figuur 28: *opsi-configed*: Tab Host parameters (Server- and Client configuration)

3.3.16 Depot configuration

In the mode *Properties of depots* you will see the tab *Depots*. There is a drop down menu to select the depot. After selecting the depot you may change the properties of the *opsi-depot*.

see also:



Figuur 29: *opsi-configed*: Tab Depot configuration

3.4 Tool: opsi-package-manager: (de-)installs opsi-packages

The `opsi-package-manager` is used for (de-)installing opsi-product-packages on an opsi-server.

In order to install a opsi-product-package, this opsi-product-package must be readable for the opsi system user opsiconfd. Therefore it is strongly recommended to install those packages from the directory /home/opsiproducts (or a sub directory).

The log file of the *opsi-package-managers* you will find at */var/log/opsi/package.log*.

Install a package (asking no questions):

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

Install a package (asking questions):

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

Install a package (and switch required action to setup where installed):

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

Deinstall a package (asking no questions):

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

Extract and rename a package:

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

Calling `opsi-package-manager` with option `--help` gives a listing of possible options.

Please note:

- The option `-d` or `--depots` are reserved for the use in a multi-depot-server environment.
- Using option `-d` the opsi-package will be copied to the `/var/lib/opsi/repository` directory of the target server before installing. Please make sure that there is enough free space on this file system.

see also:

```
#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
```

3.5 Tool: `opsi-product-updater`

The command line utility `opsi-product-updater` is designed to download and install comfortable opsi packages from a repository or a other opsi server. Using the opsi-product-updater make it easy to keep the opsi server up to date. It may be also used in a cronjob to keep depot server in sync with the config server.

```
# 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
```

The main features are:

- configurable repositories
- configurable actions

All configuration will be done at the configuration file `/etc/opsi/opsi-product-updater.conf`.

3.5.1 configurable repositories

Repositories are the sources which will be used by the opsi-product-update to fetch new opsi packages

There are two kinds of repostories:

Internet Repositories

Example: download.uib.de
This are repositories which are configured by:

- baseURL (z.B. http://download.uib.de)
- dirs (A list of directories e.g.. opsi4.0/produkte/essential)
- and if needed username and password for password protected repositories (e.g. for the opsi patch management subscriptions)

You may also configure a proxy here.

opsi-server

This is (using a *opsi-depotserver*) the central *opsi-configserver* will be used to fetch the opsi-packages.

The central configuration item is here:

- *opsiDepotId*

This in most cases on a a *opsi-depotserver* the central *opsi-configserver*. So on any call of the *opsi-product-updater* the *opsi-product-packages* wil be fechted from the *opsi-configserver*. This can be done for example by a cronjob.

3.5.2 configurable actions

For each repository you have to configure which actions to run:

- *autoupdate*: Newer versions of installed packages will be downloaded and installed
- *autoinstall*: Also packages which are not installed yet, will be downloaded and installed
- *autoinstall*: For all new installed packages and all clients on which these pacakages are installed the action request will be set to setup.

In addition it is possible to send all these clients a Wake-On-LAN signal to install the new software to the clients. Using the opsi-product *shutdownwanted* you can make shure that the clients will be powered off after the installation.

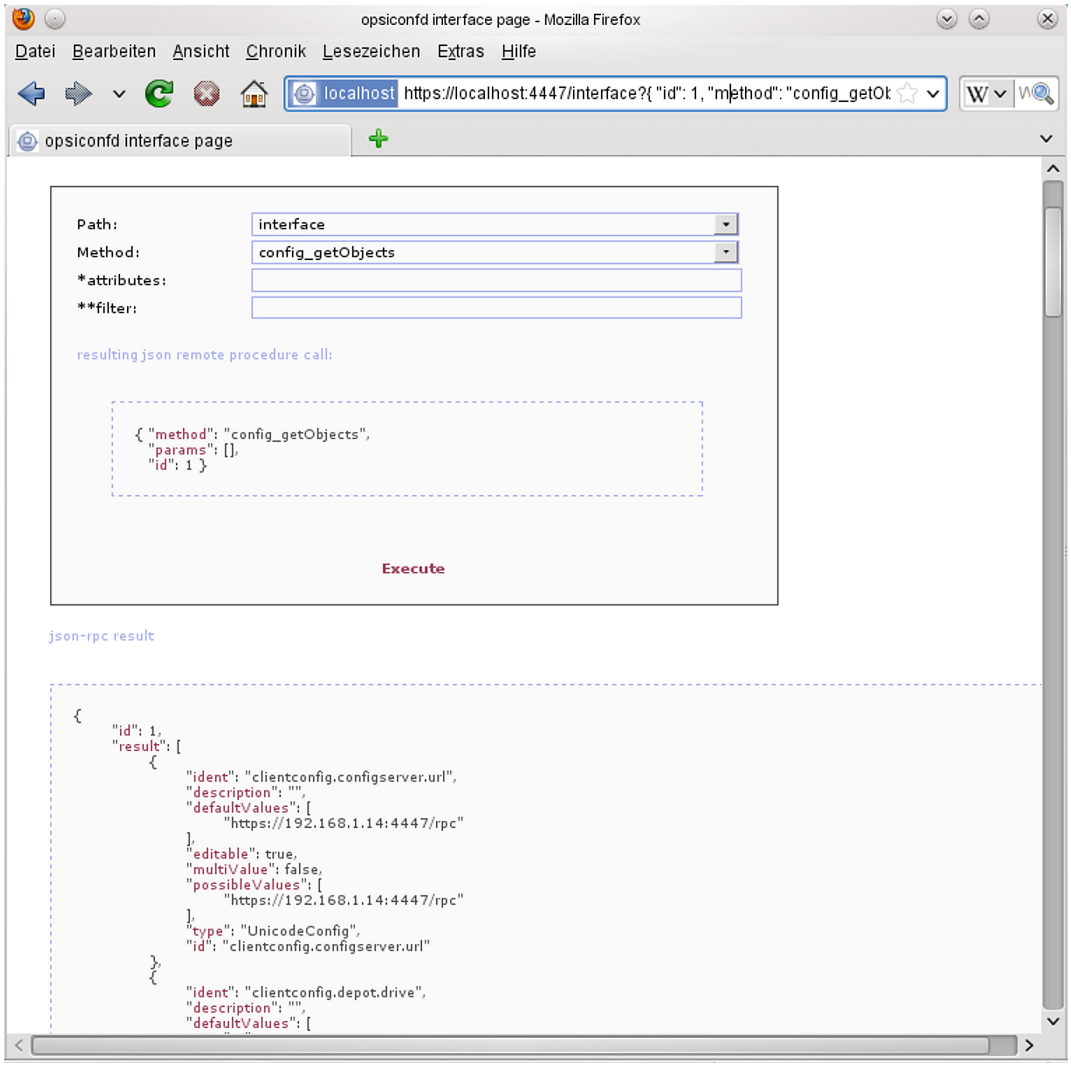
- time window for autosetup: You can give time window which may be used to that client action requests to setup.
- Automatic WakeOnLan with shutdown: If there is new software Clients could be waked up and shutdown after installation automatically

3.6 Tools: opsi-admin / *opsi config interface*

3.6.1 Overview

opsi V3 introduced an opsi owned python library which provides an API for opsi configuration. The *opsiconfd* provides this API as a web service, whereas *opsi-admin* is the command line interface for this API.

Calling https://<opsi-server>:4447/interface in your browser gives you agraphical interface to the *opsi web service*. You have to login as a member of the unix group *opsiadmin*.



Figuur 30: opsi config interface: Access to the web service via browser

At the command line *opsi-admin* provides an interface to the opsi-API. There is a interactive mode and a non interactive mode for batch processing from within scripts.

The help option `opsi-admin --help` shows a list of available command line options:

```
# 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 can use the opsi web service or directly operate on the data backend. To work with the web service you have to provide the URL and also an *username* and *password*. Due to security reasons you probably wouldn't like to do this from within a script. In that case you'd prefer direct access to the data base using the `-d` option: `opsi-admin -d`.

In interactive mode (start with `opsi-admin -d` or `opsi-admin -d -i -c` or short `opsi-admin -dic`) you get input support with the TAB-key. After some input, with the TAB-button you get a list or details of the data type of the next expected input.

The option `-s` or `-S` generates an output format which can be easily parsed by scripts.

There are some methods which are directly based on API-requests, and there are some *tasks*, which are a collection of function calls to do a more complex special job.

3.6.2 Typical use cases

Set a product to setup for all clients which have this product installed

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

List of all clients

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

Client delete

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

e.g..:

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

Client create

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

e.g.:

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

Set action request

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

e.g.:

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

Attach client description

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

set pcpatch password

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

Set the password of user pcpatch for Unix, samba and opsi.

3.6.3 Web service / API methods

Methods since opsi 4.0

In opsi 4 the data structure of all backends and the web service methods are completely new designed.

The new design is object / database oriented. A Object has some properties.

As a example let us have a look at the object *product*. A object of the type *product* which describes the product *javavm* may look like this:

```
"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": ""
```

Every object has a set of operators which ćan be used to work with this obect. Most time these operators are:

- *getObjects* (returns the objects)
- *getHashes* (Variant, which delivers for performance reasons the backend objects readonly. For a large count of objects this method is much faster then calling *getObjects*)
- *create* (create one object comfortable)
- *createObjects* (create one or more objects)

- *delete* (delete one object)
- *deleteObjects* (delete one or more objects)
- *getIdents* (returns the object id's)
- *insertObject* (create a new object)
- *updateObject* (update a object, if the object doesn't exists it will be created)
- *updateObjects* (update a bundle of objects)

The method names are concatenated:

<object name>_<operation>

According to this naming rule, these new methods are easily to difference from the old *legacy* opsi 3 methods, which almost start with *get*, *set* or *create*.

The *getObjects* methods have two optional parameters:

- *attributes*
- *filter*

The *attributes* parameter is used query only for some properties of an object. If you are using attributes the returned object has all attribute keys, but only values the attribute you asked for and for all attributes which are used to identify this object. All other attributes have the value *none*.

For Example you will get by calling the method *product_getObjects* with *attributes:["name"]* for the product *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
```


If you like to not ask for attributes but want to use the second parameter *filter* you have to give as attribute parameter *[]*.

The parameter filter is used to define which objects you want to get. For example if you are using the filter *{ ïd":"javavm"}* on the method *product_getObjects* you will get only the object(s) which describe the product *javavm*.

If you are using methods which expecting one ore more objects, these objects have to be given as JSON objects or as array of JSON objects.

The most important objects are:

- *auditHardwareOnHost* (client specific hardware information)
- *auditHardware* (client independent hardware information)

- *auditSoftwareOnClient* (client specific software information)
- *auditSoftware* (client independent software information)
- *auditSoftwareToLicensePool* (license management)
- *configState* (administration of client host parameters)
- *config* (administration of host parameter defaults)
- *group* (group administration)
- *host* (server and clients)
- *licenseContract* (license management)
- *licenseOnClient* (license management)
- *licensePool* (license management)
- *objectToGroup* (group administration)
- *productDependency* (product dependencies)
- *productOnClient* (client specific information to a product e.g. installation state)
- *productOnDepot* (depot specific information to a product)
- *productPropertyState* (depot or client specific product property settings)
- *productProperty* (definition of product properties)
- *product* (product meta data)
- *softwareLicenseToLicensePool* (license management)
- *softwareLicense* (license management)

In addition to the described objects and methods there are some more for special operations.

This design:

- is created for fast transmitting information about a lot of clients
- filter data by a unified syntax
- allows to check all input for correct synatx

According to these facts we get a increased stability and performance.

opsi3-Methoden

These methods are still available as *legacy methods*, which means that calls to these methods are mapped to the new methods internally.

Here comes a short list of some methods with a short description. This is meant mainly for orientation and not as a complete reference. The short description does not necessarily provide all information you need to use this method.

```
method addHardwareInformation hostId, info
```

Adds hardware information for the computer <hostid>. The hash <info> is passed. Existing information will be overwritten for matching keys. Applicable for special keys only.

```
method authenticated
```

Prove whether the authentication on the server was successful.

```
method checkForErrors
```

Test the backend for consistency (only available for file backend by now).

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

Creates a new client.

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

Creates a group of clients (as used by the opsi-Configed).

```
method createLicenseKey productId, licenseKey
```

Assigns an (additional) license key to the product <productId>.

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

Creates a new localBoot product (opsi-winst product).

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

Creates a new netBoot (boot image) product.

```
method createOpsiBase
```

For internal use with the LDAP-backend only.

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

Creates a new product.

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

Creates product dependencies.

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

Creates product properties.

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

Creates a new server in the LDAP-backend.

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

Not implemented yet – for future use.

```
method deleteClient clientId
```

Deletes a client.

```
method deleteGeneralConfig objectId
```

Deletes a client configuration or domain configuration.

```
method deleteGroup groupId
```

Deletes a client group.

```
method deleteHardwareInformation hostId
```

Deletes all hardware information for the computer <hostid>.

```
method deleteLicenseKey productId, licenseKey
```

Deletes a license key for product <productId>.

```
method deleteNetworkConfig objectId
```

Deletes network configuration (for example depot share entry) for a client or domain.

```
method deleteOpsiHostKey hostId
```

Deletes a pckey from the pckey data base.

```
method deleteProduct productId
```

Deletes a product from the data base.

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

Deletes product dependencies.

```
method deleteProductProperties productId *objectId
```

Deletes all properties of a product.

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

Deletes a single product property.

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

Deletes a single property or all properties from the product <productId>.

```
method deleteServer serverId
```

Deletes a server configuration

```
method exit
```

Quit the *opsi-admin*.

```
method getBackendInfos_listOfHashes
```

Supplies information about the available backends of the opsi depot server and which of them are activated.

```
method getBootimages_list
```

Supplies the list of the available boot images.

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

Supplies a list of clients which meet the assigned criteria.

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

Supplies an extended list of clients which meet the assigned criteria (with description, notes and *last seen* for each client).

```
method getDefaultNetBootProductId clientId
```

Supplies the netboot product (for example: system software) which will be installed when the boot image *install* is assigned.

```
method getDomain hostId
```

Supplies the computer domain.

```
method getGeneralConfig_hash objectId
```

Supplies the general configuration of a client or a domain.

```
method getGroupIds_list
```

Supplies the list of saved client groups.

```
method getHardwareInformation_listOfHashes hostId
```

Supplies the hardware information of the specified computer.

```
method getHostId hostname
```

Supplies the hostid of the specified host name.

```
method getHost_hash hostId
```

List of properties of the specified computer.

```
method getHostname hostId
```

Supplies the host name of the specified host id.

```
method getInstallableLocalBootProductIds_list clientId
```

Supplies a list of all localBoot products that could be installed on the client.

```
method getInstallableNetBootProductIds_list clientId
```

Supplies a list of all netBoot products that could be installed on the client.

```
method getInstallableProductIds_list clientId
```

Supplies a list of all products that could be installed on the client.

```
method getInstalledLocalBootProductIds_list hostId
```

Supplies a list of all localBoot products that are installed on the client.

```
method getInstalledNetBootProductIds_list hostId
```

Supplies a list of the installed netBoot products of a client or server.

```
method getInstalledProductIds_list hostId
```

Supplies a list of the installed products for a client or server.

```
method getIpAddress hostId
```

Supplies the IP address of a host.

```
method getLicenseKey productId, clientId
```

Supplies an available license key of the specified product or the product license key which is assigned to the client.

```
method getLicenseKeys_listOfHashes productId
```

Supplies a list of all license keys for the specified product.

```
method getLocalBootProductIds_list
```

Supplies a list of all (for example in the LDAP-tree) known localBoot products.

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

Supplies for all clients the installation status and action request of all localBoot products.

```
method getMacAddresses_list hostId
```

Supplies the MAC address of the specified computer.

```
method getNetBootProductIds_list
```

Supplies a list of all NetBoot products.

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

Supplies for all clients the installation status and action request of all netBoot products.

```
method getNetworkConfig_hash objectId
```

Supplies the network specific configurations of a client or a domain.

```
method getOpsiHostKey hostId
```

Supplies the pckey of the specified hostid.

```
method getPcpatchPassword hostId
```

Supplies the password of *pcpatch* (encrypted with the *pckey* of *hostId*).

```
method getPossibleMethods_listOfHashes
```

Supplies the list of callable methods (approximately like in this chapter).

```
method getPossibleProductActionRequests_list
```

Lists the available action requests of opsi.

```
method getPossibleProductActions_hash
```

Supplies the available actions for each product (*setup*, *deinstall* , ...).

```
method getPossibleProductActions_list productId=softprod
```

Supplies the list of all actions (*setup*, *deinstall*,. . . .).

```
method getPossibleProductInstallationStatus_list
```

Supplies the list of all installation states (*installed*, *not_installed*,. . .)

```
method getPossibleRequirementTypes_list
```

Supplies the list of types of product requirement (*before*, *after*, . . .)

```
method getProductActionRequests_listOfHashes clientId
```

Supplies the list of upcoming actions of the specified client.

```
method getProductDependencies_listOfHashes productId = None
```

Supplies the list of product dependencies of all or the specified product.

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

Supplies a list of products which meet the specified criteria.

```
method getProductInstallationStatus_hash productId, hostId
```

Supplies the installation status for the specified client and product.

```
method getProductInstallationStatus_listOfHashes hostId
```

Supplies the installation status of the specified client.

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

Supplies the product properties of the specified product and client.

```
method getProductPropertyDefinitions_hash
```

Supplies all known product properties with description, allowed values,. . .

```
method getProductPropertyDefinitions_listOfHashes productId
```

Supplies the product properties of the specified product with description, allowed values,. . . .

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

Supplies installation status and action requests of all products (for the specified clients).

```
method getProduct_hash productId
```

Supplies the meta data (description, version, . . .) of the product

```
method getProvidedLocalBootProductIds_list serverId
```

Supplies a list of available localBoot products on the specified server.

```
method getProvidedNetBootProductIds_list serverId
```

Supplies a list of available netBoot products on the specified server.

```
method getServerId clientId
```

Supplies the opsi-configserver in charge of the specified client.

```
method getServerIds_list
```

Supplies a list of the known opsi-configserver.

```
method getServerProductIds_list
```

Supplies a list of the server products.

```
method getUninstalledProductIds_list hostId
```

Supplies the products which are uninstalled.

```
method powerOnHost mac
```

Send a WakeOnLan signal to the specified MAC address.

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

Set a *bootimage* for the specified client.

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

Set for client or domain the generalConfig

```
method setHostDescription hostId, description
```

Set a description for a client.

```
method setHostLastSeen hostId, timestamp
```

Set the *last seen* time stamp of a client.

```
method setHostNotes hostId, notes
```

Set the notes for a client.

```
method setMacAddresses hostId, macs
```

Set the client MAC address in the data base.

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

Set the specified network data for the opsi-client-agent for a client.

```
method setOpsiHostKey hostId, opsiHostKey
```

Set the *pckey* for a computer.

```
method setPXEBootConfiguration hostId *args
```

Set the pipe for PXE-Boot with *args in the *append*-List.

```
method setPcpatchPassword hostId password
```

Set the encrypted (!) *password* for *hostId*

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

Set an action request for the specified client and product.

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

Set an installation status for the specified client and product.

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

Set the product properties for the specified product (and the specified client).

```
method unsetBootimage hostId
```

Unset the boot image start for the specified client.

```
method unsetPXEBootConfiguration hostId
```

Delete PXE-Boot pipe.

```
method unsetProductActionRequest productId, clientId
```

Set the action request to *none*.

Backend extensions

In opsi 4 is we have the possibility to extend the basic opsi 4 methods with own additional methods which use the opsi 4 base methods. This is done for example to implement the opsi 3 legacy methods or to create methods which fits better to the needs of the opsi-configed.

These extenstions has to be written as Python code in the */etc/opsi/backendManager/extend.d* directory.

3.7 Server processes: opsiconfd and opsipxeconfd

The *opsipxeconfd* provides the *named pipes* in the `tftpboot` directories. which are used to control the PXE boot process.

The configuration file is `/etc/opsi/opsipxeconfd.conf`

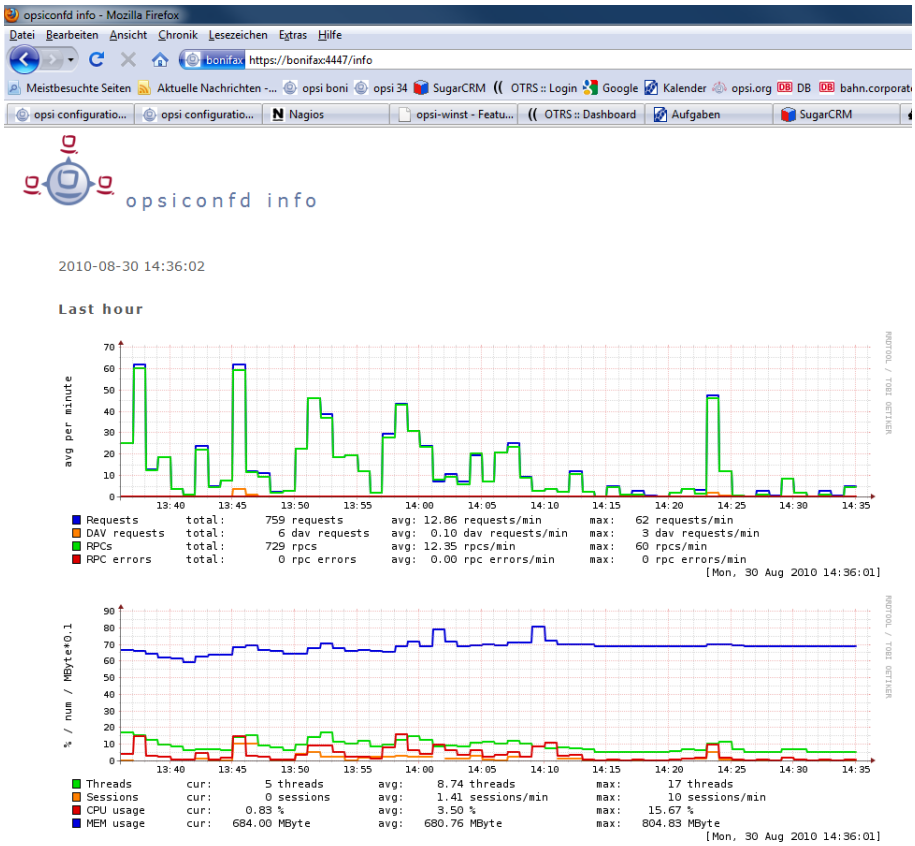
The log file is `/var/log/opsi/opsipxeconfd.log`.

The *opsiconfd* provides the opsi API as JSON web service and have a lot of other important tasks. Therefore the *opsiconfd* is the central opsi service and does all the communication to the clients.

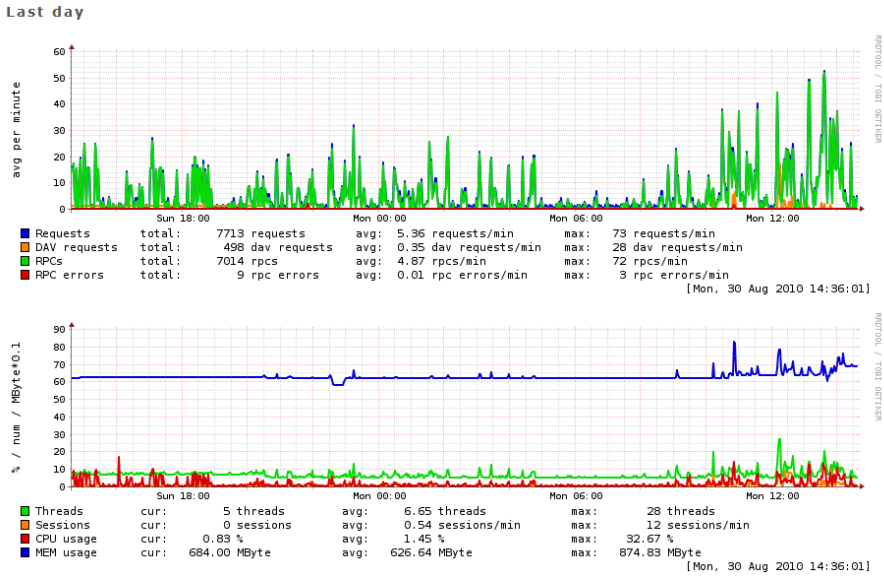
Regarding this central rule, a tool to monitor this process gives a lot of information about load and possible problems. This tool is the *opsiconfd* info page.

3.7.1 *opsiconfd* monitoring: opsiconfd info

Using the web address https://<opsi-server>:4447/info you will get a graphical chart of *opsiconfd* load and cpu/memory usage in the last hour/day/month/year. This information is completed by tabulary information to the actual tasks and sessions.



Figuur 31: opsiconfd info: opsiconfd values from the last hour



Figuur 32: opsiconfd info: opsiconfd values from the last day

4 Activation of non free modules

Even opsi is open source, there are some components which are not free at the moment. At this time (May 2011) the following components of opsi are not free:

- license management
- the MySQL backend for configuration data
- the support for hierarchical client groups
- WAN/VPN extension
- high availability and load balancing (not implemented yet)
- Software on Demand

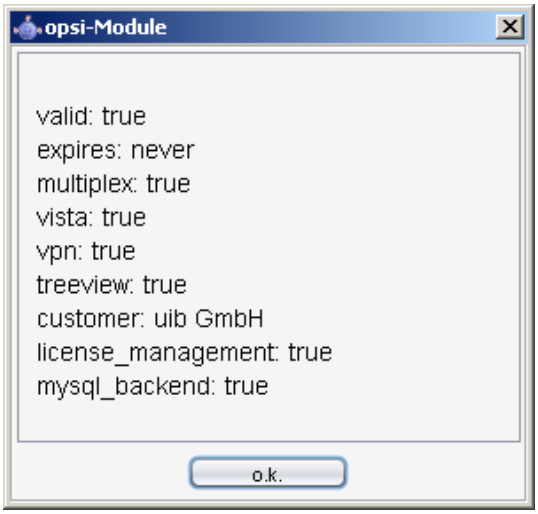
These components are developed in a co-funding project which means that until the complete development costs are payed by co-funders, they are only allowed to use by the co-funders or for evaluation purposes. If we have earned the development cost we will give these modules for everybody for free. To control the use of these components until they are free there is a activation file `/etc/opsi/modules`, which is protected against changes via electronic signature. If this activation file doesn't exist, only the *free* parts of opsi will work.

If you need for evaluation a temporary valid activation file please contact info@uib.de. If you become a co-funder, you will get a unlimited activation file. Copy this file as *root* to `/etc/opsi/modules`. If this is done, execute:

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

You may check your activation state with one of the following methods:

Using the *opsi-configed* choose the menu entry *Help/opsi-Module* which shows a window with the activation state.



Figuur 33: Display of activation state in opsi-configed

At the command line you may use the command opsi-admin with the method `backend_info`. (Remark: Never give your activation file or the output of this command to third people without deleting the signature).

```
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
}
}
```

5 opsi-client-agent

5.1 Overview

To make Software distribution manageable for the system administrator, a client computer has to notice that new software-packets or updates are available and install them without user interaction. It is important to make user-interaction completely obsolete as the installation can run unattended this way and a user cannot stop the installation during the installation process.

These requirements are implemented in opsi by the *opsi-client-agent*:

On the client side the service *opsiclientd* usally at boot time before the user logs in examines whether an update has to be installed for this client.

If there are software to be installed on the client, the script processing program *opsi-winst* is being started to do the installation job. The server provides all the installation scripts and software files on a file share. At this time the user has no chance to interfere with the installation process.

As an additional option the module *loginblocker* can be installed to prevent a user login before the end of the installation process is reached.

Before a software can be installed with the *opsi-winst* program, they have to be prepared as *opsi-product-package*. For details see Chapter *Integration of new software packets into the opsi software deployment* at the *getting started* manual.

5.2 Directories of the opsi-client-agent

The *opsi-client-agent* is installed at `%ProgramFiles%\opsi.org\opsi-client-agent`.

This directory contains all programs of the *opsi-client-agent* like e.g. the *opsiclientd*, the *opsiclientd notifier* the *opsi-winst* and some needed libraries. Also we will find here the configuration files and graphical templates (skins) of the mentioned programs.

The directory `%ProgramFiles%\opsi.org\opsi-client-agent` is protected against manipulation by users without administartor priviliges.
The directory `%ProgramFiles%\opsi.org\opsi-client-agent\opsiclientd` contains the configuration file of the *opsiclientd* and you need administrator privilges to read it.

There is a other directory `c:\opsi.org`.

This directory is used (at the moment) for caching installation files and data (see WAN-Extension). In future it will have some more function like containig log files.
You need administrator privilges to read the directory `c:\opsi.org`.

The log files of the *opsi-client-agent* you will find at `c:\tmp`.

5.3 The service: opsiclientd

The *opsiclientd* is the core of the *opsi-client-agent*. The *opsiclientd* starts at boot time and runs with administrative privileges.

The important features are:

- Event based control:
  The activity of the opsi client agent (opsiclientd) may be triggered by different events in the client system. According to this fact the start of the installation is not fixed at the system start up any more.
- Control via web service:
  This interface is used for *push* installations and for maintenance purpose as well.
- Remote configuration:
  The configuration data for the clients may be changed (globally or client specific) at the server by editing the *Host parameter*.

The *opsi-client-agent* consists of multiple components :

- *opsiclientd*: the central service
- *opsiclientd notifier*: information and communication window
- *opsi-login-blocker*: block the login until the installation has finished

5.3.1 Installation

In case of automatic OS-Installation with opsi (not image based), the *opsi-client-agent* will be installed automatically.

You may set the action request *unstall* to uninstall the *opsi-client-agent*.

For a subsequent installation on a existing Windows system or for repair purposes see at the *getting started* manual.

5.3.2 opsiclientd

Core component of the *opsi-client-agent* is the service *opsiclientd*. This service starts at the boot time.

The *opsiclientd* has the following tasks:

- while the system is booting and the *opsiclientd* is waiting for the GUI to come up the *block_login_notifier* is started wich shows a padlock at the righit upper corner of the screen.
- Getting active if the configuration event takes place. If it geat active the *opsiclientd* contacts the opsi server via web service (JSON-RPC) and ask for configuration data and required actions.
  The default event is *gui_startup* which will fire at boot time and before login.
- Creates a named pipe which is used by the *opsi-login-blocker* to ask via JSON-RPC the *opsiclientd* when to unblock the login.
- Starting the *opsiclientd notifier* as thread for information and interaction with the user.
- If needed, it connects to the *opsi-depot*, update and start of the *opsi-winst* and start this program to process the *action requests* (installations).

5.3.3 opsiclientd notifier

The *opsiclientd notifier* implements the interaction with the user. They displays status messages and may give the possibility to interact with the process.

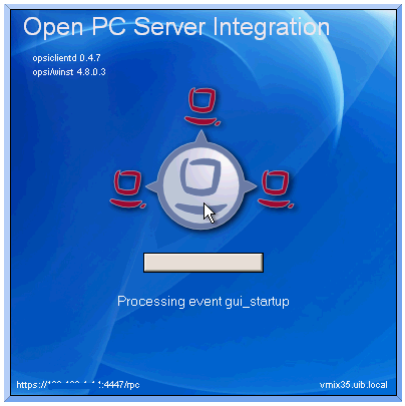
There are different situations where the *opsiclientd notifier* will become active in different ways:

blocking notifier
Indicates that the *opsi-login-blocker* is blocking

opsiclientd blocklogin notifier

Figuur 34: opsiclientd blocklogin notifier

event notifier
Give information to a started event.



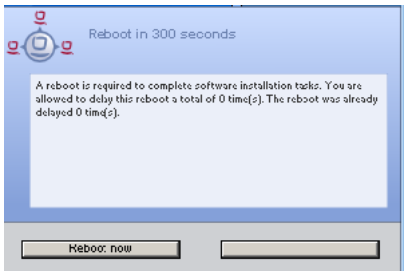
Figuur 35: opsiclientd event notifier

action notifier
Shows state of the event processing

opsiclientd action notifier

Figuur 36: opsiclientd action notifier

shutdown notifier
Gives information about a requested reboot / shutdown (if shutdown_warning_time > 0)



Figuur 37: opsiclientd shutdown notifier

Let op
The names and functionality of the notifier has changed from opsi 4.0 to opsi 4.0.1.
The opsi 4.0 event notifier doesn't exist's anymore.
The opsi 4.0.1 event notifier is equal to the opsi 4.0 action notifier.
The opsi 4.0.1 action notifier has nearly the same functionality like the opsi 4.0 event notifier, but it will only be activated if there is a *action request*.

5.3.4 opsi-login-blocker

Der *opsi-login-blocker* für NT5 Win2K/WinXP ist als *GINA* implementiert (*opsigina.dll*). Diese *GINA* wartet bis zum Abschluss der *product actions* 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-login-blocker* für NT6 (Vista/Win7) ist als *credential provider filter* realisiert (*OpsiLoginBlocker.dll*). Er blockiert alle *credential provider* bis zum Abschluss der *product actions* oder dem Timeout (Standard-Wert: 120 Sekunden) bei nicht erreichbarem *opsiclientd*.

5.3.5 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: Ablauf eines Standard-Events

Figuur 38: Ablauf eines Standard-Events

Die wichtigsten Parameter wirken hier wie folgt zusammen:

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_cancellable_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_cancellable_after` auf einen Wert größer oder gleich `connection_timeout` gesetzt werden.

Tip
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. 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 `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.

2. 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 standardmäßig startende *opsiclientd notifier* verhält sich analog zum *opsiclientd notifier*, der die Aktionen ankündigt 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.

Tip
Der Ablauf des Event und auch die Aktionen des Benutzers sind in der Timeline auf der Info-Seite des *opsiclientds* sichtbar (siehe Paragraaf 5.3.8).

Abbildung: Vollständiges Ablaufdiagramm eines Events

Figuur 39: Vollständiges Ablaufdiagramm eines Events

5.3.6 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 ërben". 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).
-

Konfiguration über die Konfigurationsdatei

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

Let op
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

# 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 = \\install\\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%

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     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 = 300

[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 `[global]` 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: Serverweite Konfiguration des opsiclientd über den opsi-configed

Figuur 40: 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: Client spezifische Konfiguration des opsiclientd über den opsi-configed

Figuur 41: Client-spezifische Konfiguration des opsiclientd über den opsi-configed

5.3.7 Logging

Die Log-Datei des *opsiclientd* ist standardmäßig `c:\tmp\opsicliend.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-login-blockers* befindet sich unter NT6 (Vista/Win7) als auch unter NT5 (Win2k/WinXP) in `c:\tmp\opsi_loginblocker.log`.

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

Figuur 42: Info-Page des opsiclientd nach einer Push-Installation mit aktiviertem Produkt-Caching

5.3.9 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.*öder "*.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 Paragraaf 3.3.8

Auf der Kommandozeile lässt dich 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 *product actions* auszuführen.

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

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-keys* als Passwort.

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

Abbildung: Webservice des opsiclientd

Figuur 43: 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
```

5.4 Sperrung des Anwender Logins mittels opsi-login-blocker

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

Ob der *opsi-login-blocker* währen der *opsi-client-agent*-Installation installiert bzw. aktiviert wird, kann über das *product property* `loginblockerstart` konfiguriert werden.

5.4.1 opsi-login-blocker unter NT5 (Win2k/WinXP)

Der opsi-login-blocker (`opsigina.dll`) ist als *GINA* realisiert. Die *opsigina* wartet bis zum Abschluss der *product actions* 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-login-blocker 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*.

5.4.2 opsi-login-blocker unter NT6 (Vista/Win7)

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

5.5 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*.

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

has to be written