Documentation

opsi
Version 3.4

open pc server integration

Manual

Status: 6/29/10

uib gmbh
Bonifaziusplatz 1B
D - 55118 Mainz, Germany

phone: +49 - (0)6131-275610
www.uib.de
info@uib.de

Table of Contents

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 automatic software distribution system *opsi* ("**o**pen **p**c **s**erver **i**ntegration"). 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 background color grey:

```
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, education and maintenance.

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 (*GNU*-tools, *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 main 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
- management of licenses

The functionality of opsi is based on the opsi server which allocates the server-sided services.

2.3. What's new at opsi 3.4

- Management of software licenses, which isn't free yet
  (you have to pay 1 000 € once)
- opsi-confied with support for context sensitive menus (right click)
- Enhanced preloginloader 3.4 with two alternative modes:
  - 'opsiclientd' which supports Vista and Windows 7, and have some more new features but isn't free yet (you have to pay 2 000 € once)
  - 'prelogin' which is the well known and free preloginloader 3.3 technology
- Activation file to protect non free parts of opsi
  Even opsi is open source, there are some components which are not free at the moment. 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.

2.4. What you should read in case of a upgrade to opsi 3.4

At this manual.

- 4 Activation of non free modules: opsiclientd, license management, VPN-support on page 39
- 5 preloginloader 3.4 on page 41
- Fehler: Referenz nicht gefunden Fehler: Referenz nicht gefunden on page Fehler: Referenz nicht gefunden
- 8 opsi license management on page 72

3. opsi configuration and tools

3.1. Overview

The configuration of opsi requires some data management. In opsi V2 there only was a file based data management and the old tools operated directly on the files (they still can be used with the file backend). Since opsi V3 there are several types of data management backends available and new tools which are using a web service for data exchange. They exchange data via the 'opsiconfd', and the 'opsiconfd' forwards the data to the backend manager which passes the data into the selected backend. More about this is to be found in chapter 'data management of opsi'.

The default backend is the File31 backend.

3.2. Tool: opsi V3 opsi-Configed

3.2.1. Requirements and operation

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

The opsi-configed is one component of the client product 'opsi-adminutils' and can be started from the opsi-adminutils-group in the start menu.

On the server the opsi-configed will be installed as debian packet (opsi-configed.xxxxx.deb) and can be started with a menu entry in the desktop menu as well as **`/usr/bin/opsi-configed`**.

Also it 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
```

The default port is port 4447. A different port can be selected together with the host parameter, like '<host>:<port>'.

3.2.2. Login

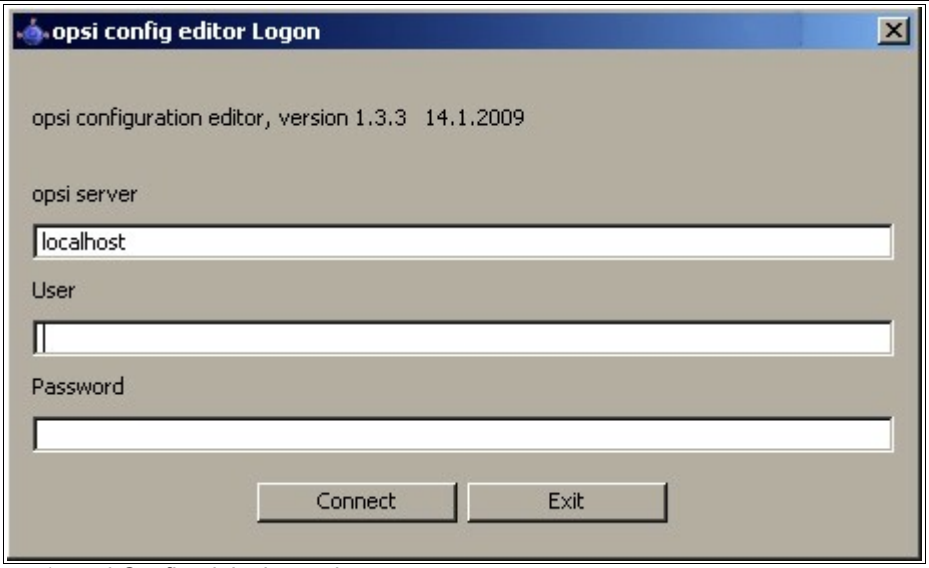


Figure 1: opsi-Configed: login mask

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 depot server account. For a successful login the provided user has to be a member of the unix-group 'opsiadmin'.

3.2.3. Depot selection

All depots integrated with your server are listed in the upper left corner of the opsi-configed. By default the depot on your opsi-config-server is selected and the clients belonging to this depot are shown. If you select multiple depots (in the usual manner of multi-item-selection in a list , eg. with shift/ctrl + click) you have to reload the data for getting any effects. If the selected server set is not synchronous (and can therefore not be handled on common grounds) your are told so. Otherwise the client list of the combined depots is shown, and their configurations may be editied.

3.2.4. Single client selection and batch selection

After a successful login the main window pops up and shows the tab 'Client selection'. This tab shows a list of known clients with the columns 'client name', 'description' 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
- 'last seen' shows the date and a time of the last client connect to the opsiconfd web service
- '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.

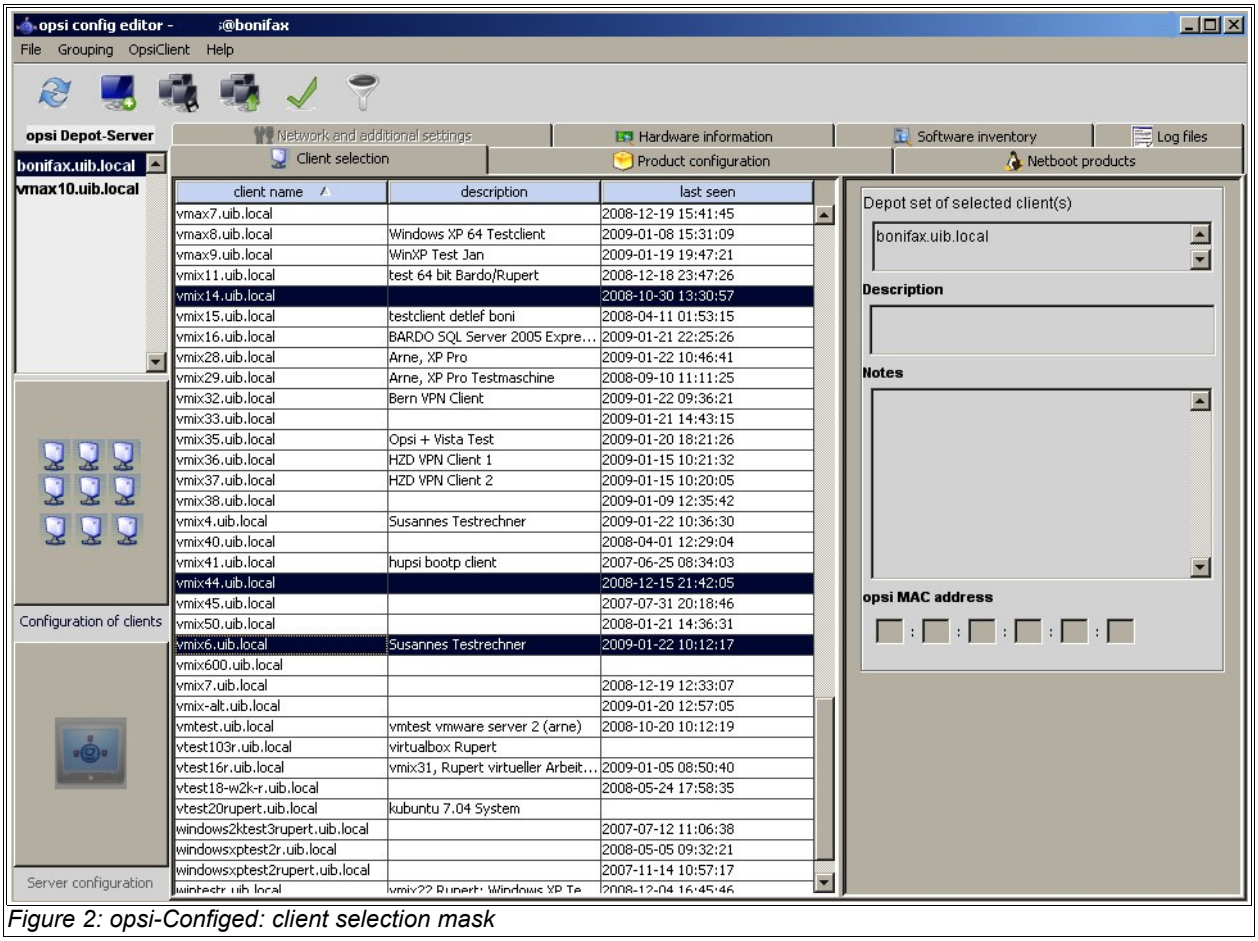


Figure 2: opsi-Configed: client selection mask

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

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.

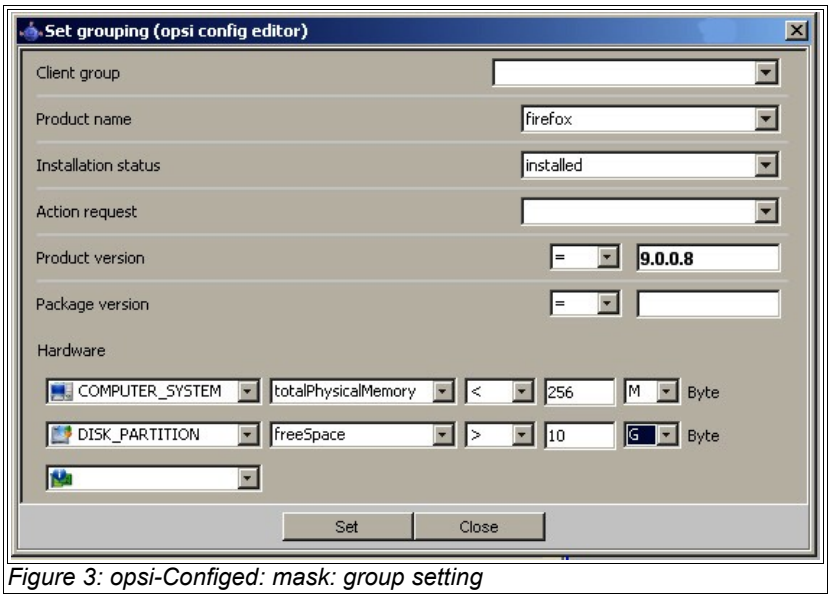


Figure 3: opsi-Configed: mask: group setting

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

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').

3.2.5. Client processing / WakeOnLan / Create a Client / Move a Client

You can select one or more clients and send them a 'WakeOnLan' signal by choosing this option from the menu 'OpsiClient'.

In the same menu you find the option for deleting selected clients, and creating a client.

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.

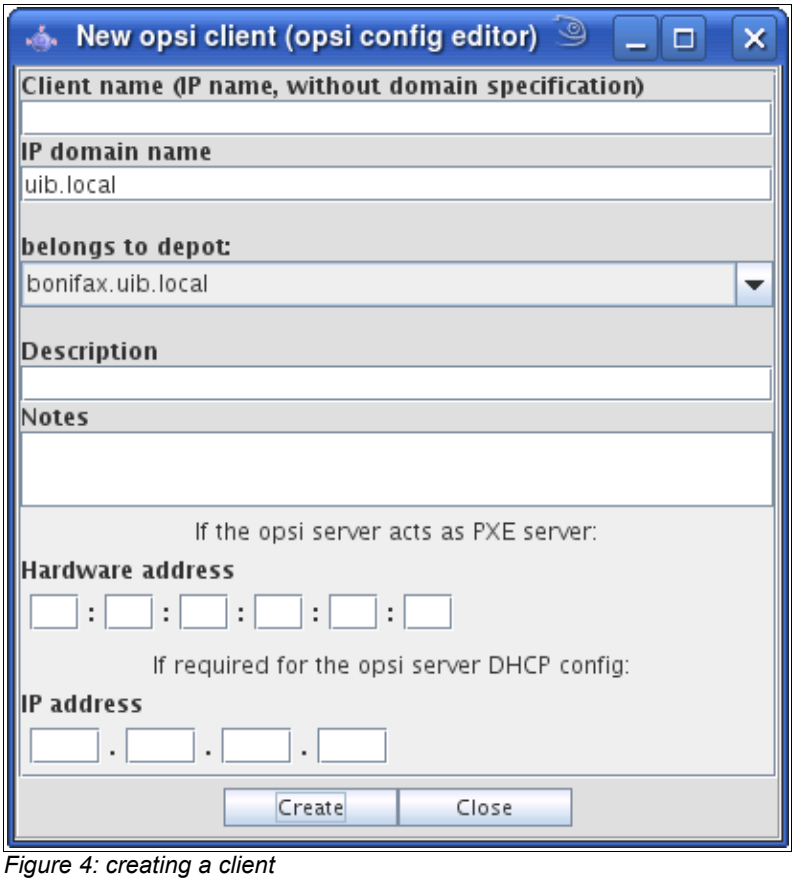


Figure 4: creating a client

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 'File31'-backend in <pcname>.ini and the IP-number will be discarded.

In version 3.3 a menu item was added for moving a client to a different depot-server. If clicked the following windows appears with a list of existing depot-servers (only supported for professional support contracts):

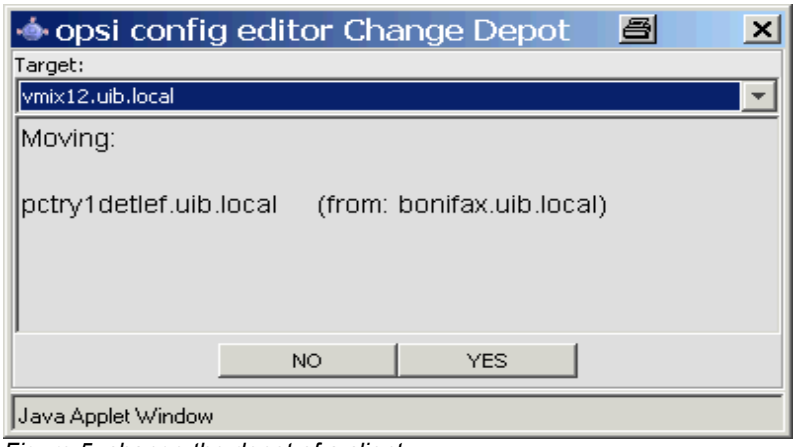


Figure 5: change the depot of a client

3.2.6. 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. 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.

- 'installation state' is the last announced state of the product and can hold the values 'installed', 'not installed', 'installing', 'undefined' and 'failed'. 'failed' means that the installation script announced an installation abort. 'Undefined' means the multiple selected clients have a different state. 'Installing' is the state during an product installation
- 'action request' is the next action to start. Possible values are 'none', 'undefined' and actions declared by the product script like: 'setup', 'deinstall', 'once', 'always'

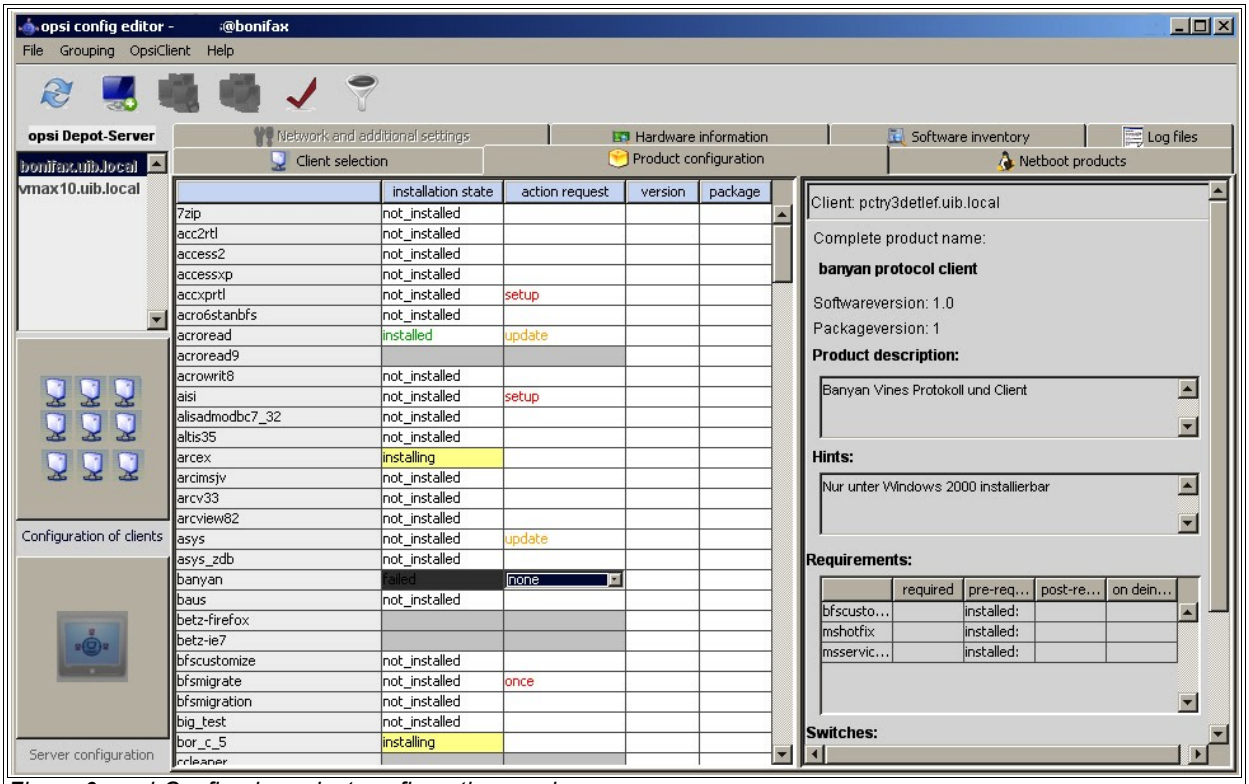


Figure 6: opsi-Configed: product configuration mask

- 'version' is the version number of the software installed on the client (as defined in the opsi packet)
- 'package' is the package number of the opsi-packet installed on the client

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 packet

'Softwareversion': software version number of the software packet (specified in the opsi installation packet)

'Packageversion': version of the packet

'Product description': free text to describe the software

'Hints': free text with advices and caveats for handling the packet

'Requirements': A list of packets which the selected product depends on and the type of dependency: 'required' means the chosen product requires that packet, but it doesn't matter whether it is installed before or after the product itself. 'pre-required' means that packet has to be installed before the product installation. 'post-required' means the packet needs to be installed after the product installation. 'on deinstall' means this action should take place before the chosen product will be de-installed.

'Switches': For a client specific configuration additional product specific switches can be defined by the product. The list of available switches is shown. The meaning of the switch is shown in the tool tip (when the cursor is moved over the switch name). Under 'property value' you get a list of permitted options for this switch. If there is no list, the packet does not provide a restricted option list and the value can be any free text.

3.2.7. 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.

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

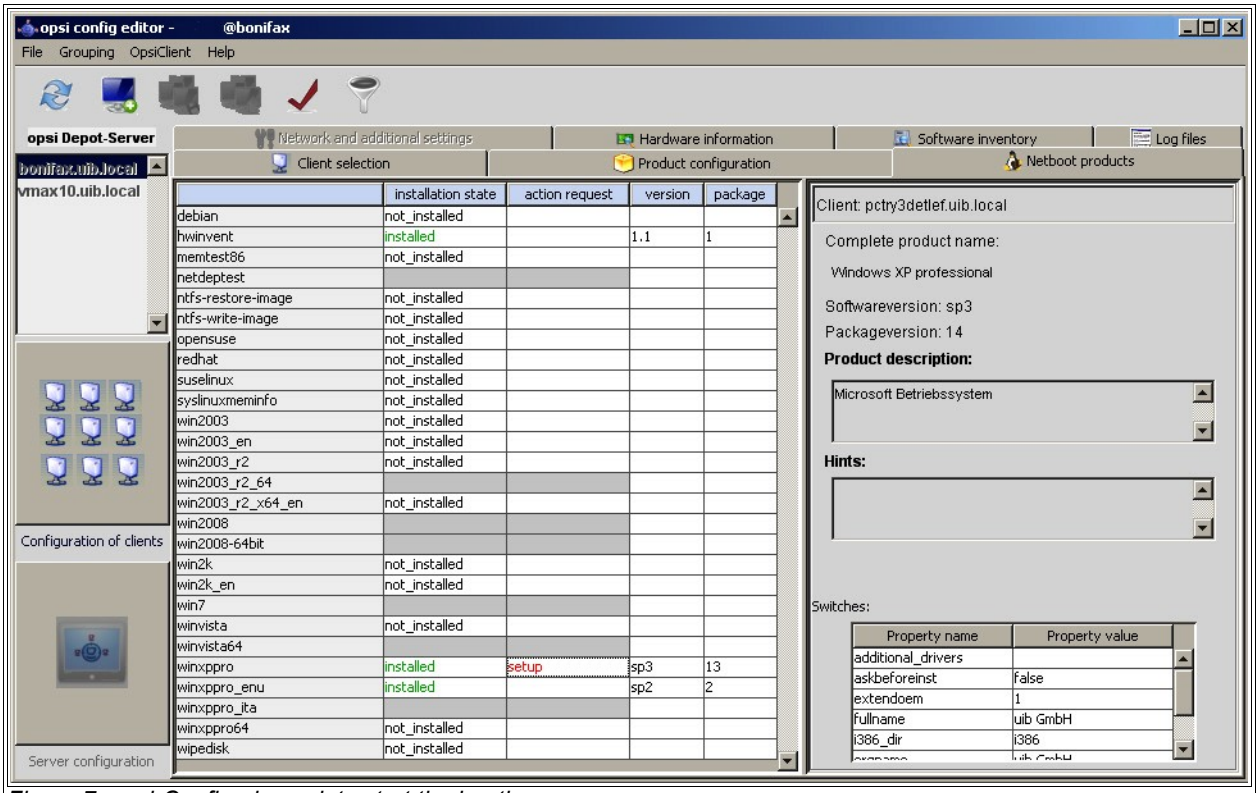


Figure 7: opsi-Configed: mask to start the bootimage

3.2.8. Hardware information

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

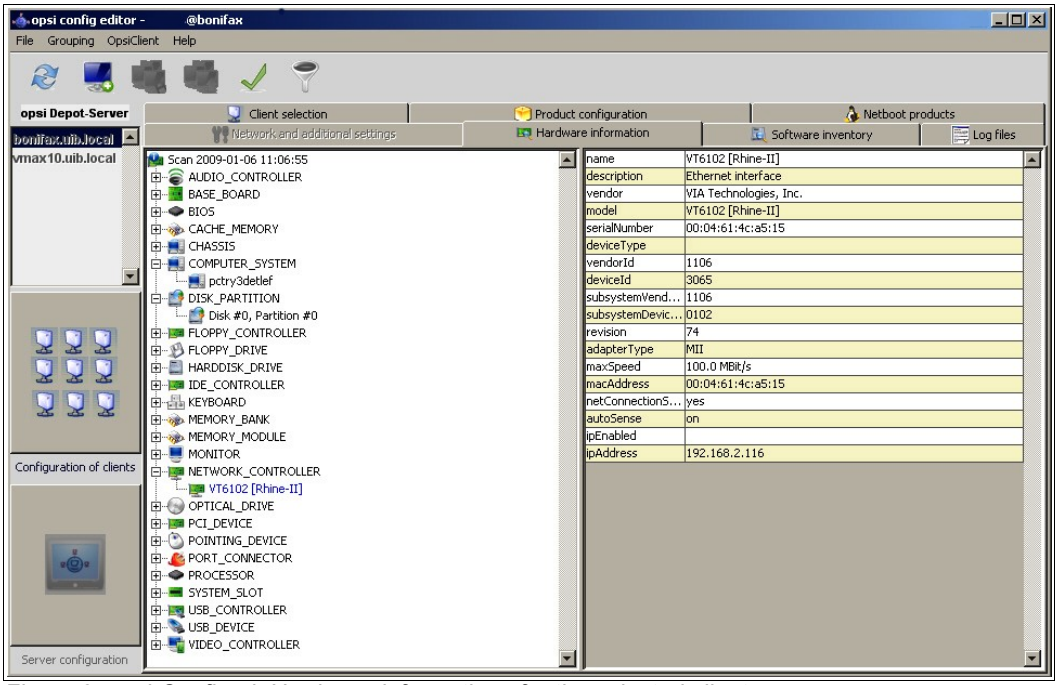


Figure 8: opsi-Configed: Hardware informations for the selected client

3.2.9. Software inventory

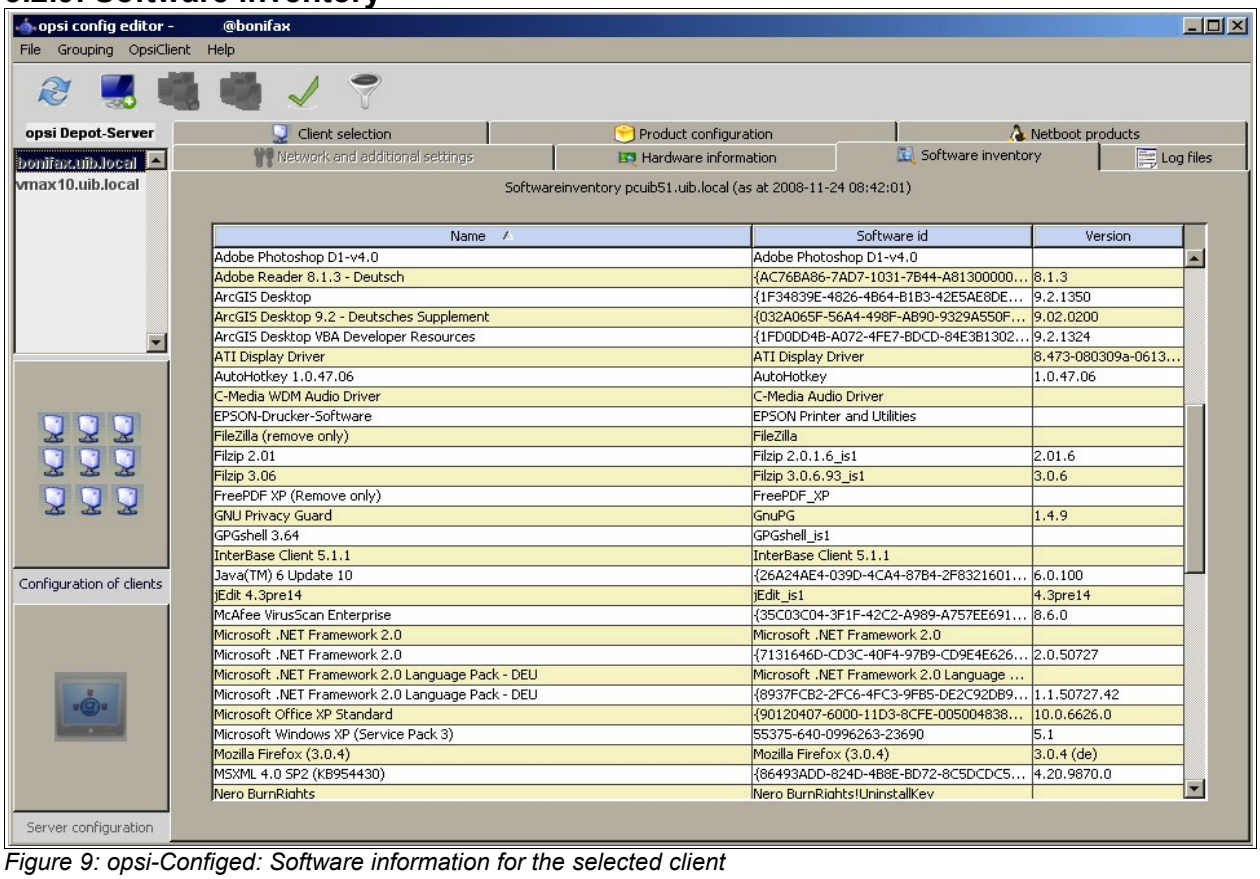


Figure 9: opsi-Configed: Software information for the selected client

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

3.2.10. Logfiles: Logs from client and server

Since opsi 3.3 the client log files are stored on the server and visible with the opsi-configed.

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

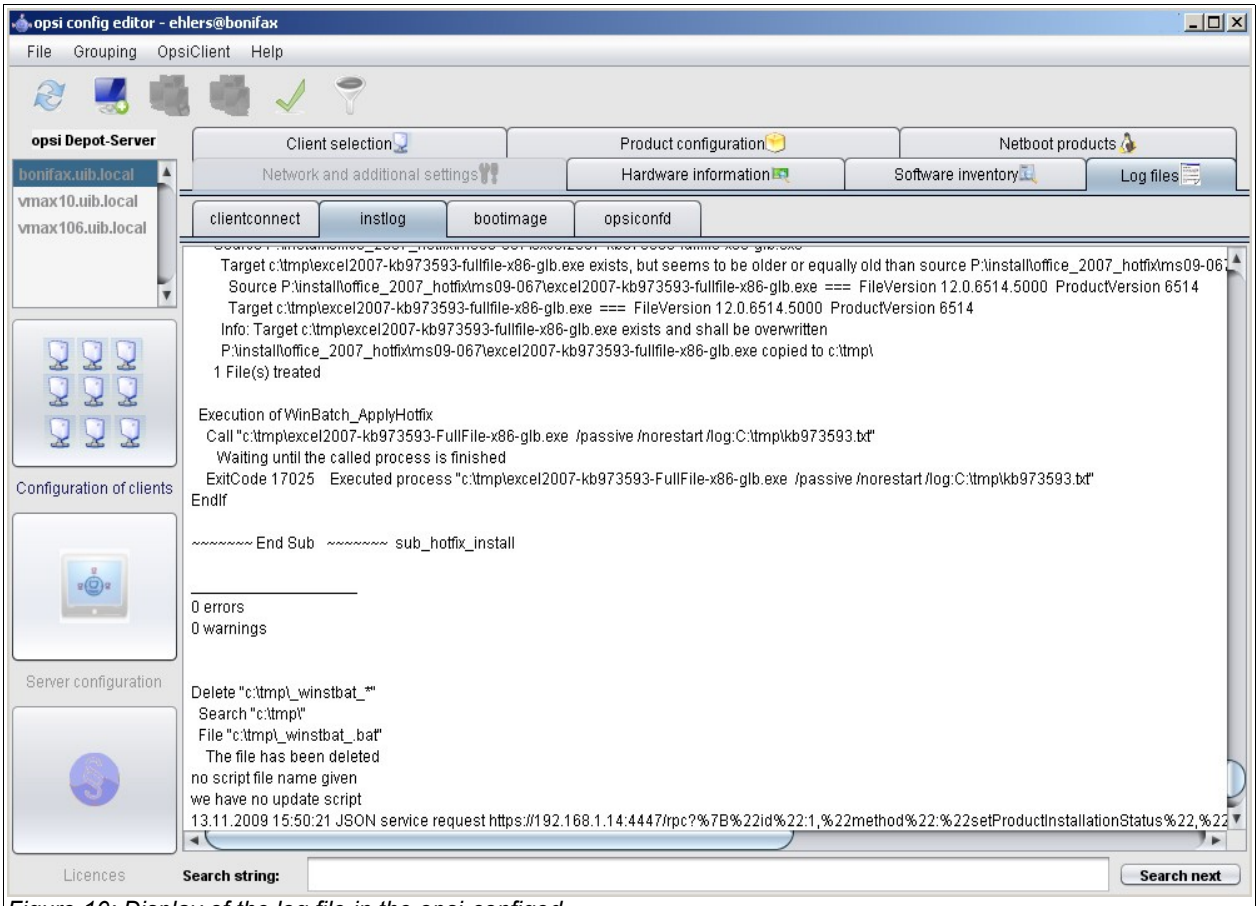


Figure 10: Display of the log file in the opsi-configed

3.2.11. Server configuration: network and additional settings

With the tab 'Network and additional settings' you can provide settings for the network configuration of opsi and other optional configurations. The options are described in chapter 6.1 “Filebackends / File31 / <pcname>.ini”.

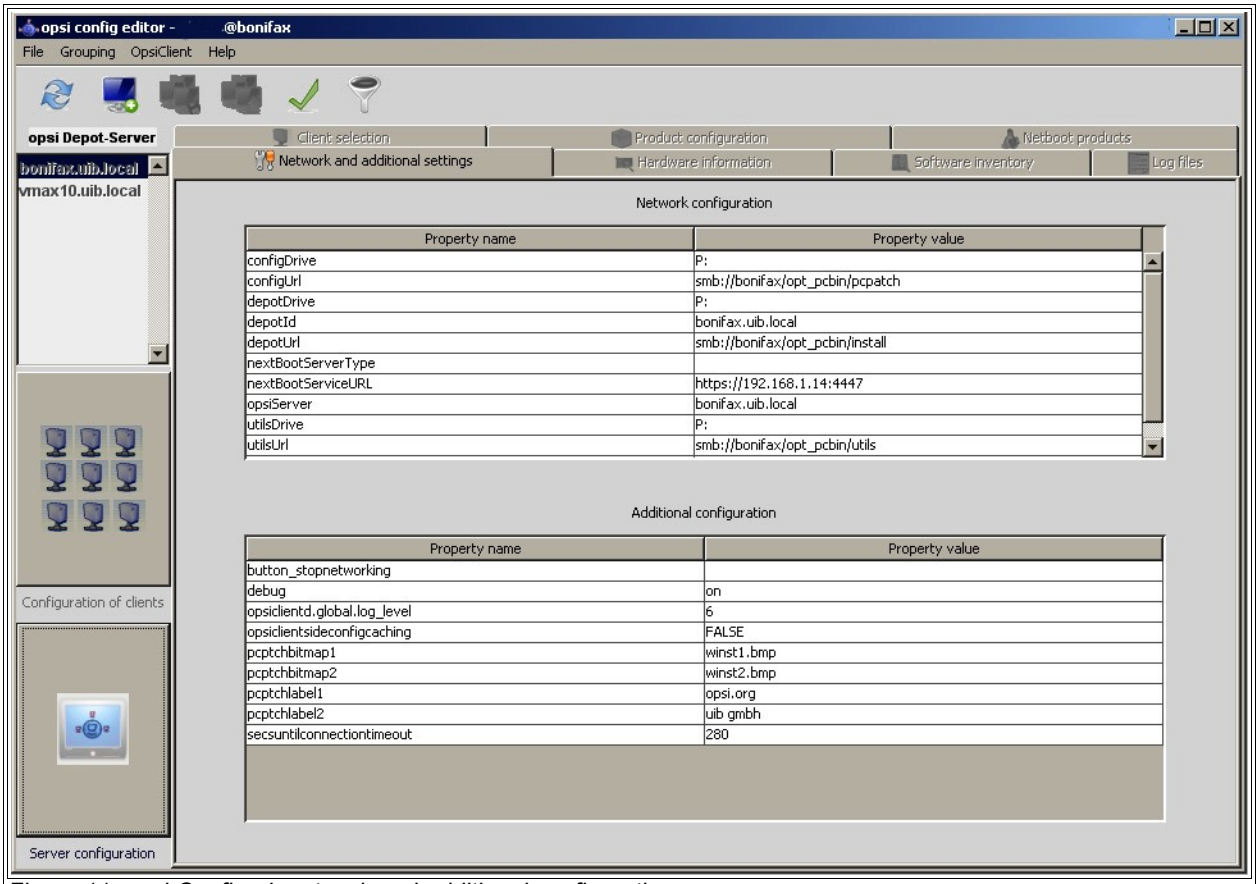


Figure 11: opsi-Configed: network and additional configuration

3.3. Tool: opsi V3 opsi-Webconfiged

The 'configed' as described above is available as an applet if the debian-packet 'opsi-configed' is installed on the server.

Start configed from a browser: http[s]://<servername>:<port>/configed/
Example: https://dpvm03:4447/configed/

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

The opsi-package-manager is used for (de-)installing opsi-packages on an opsi-server. opsi-package-manger replaces the former and now deprecated commands opsiinst and opsiuninst.

In order to install a opsi-package this opsi-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).

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-productid myprod
```

Calling opsi-package-manager with option -h gives a listing of possible options.
The option -d or --depots are reserved for the use in a multi-depot-server environment and you will get commercial support only based on a professional support contract.
Using option -d the opsi-package will be copied to the /var/lib/opsi/products directory of the target server before installing. Please make sure that there is enough free space on this file system. See also:
chapter Fehler: Referenz nicht gefunden10 opsi-server with multiple depots page 111

```
svmopside:~# opsi-package-manager -h

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:
  -d, --depots      <depots>       comma separated list of depots to process
                                   (default: <this-host>.<myDomain>)
                                   use keyword ALL to process all known depots
  --direct-install          install package directly without repository upload
  -p, --properties  <mode>         mode for default product property values
                      ask                      display dialog
                      package                  use defaults from package
                      keep                     keep depot defaults (default)
  -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"
  --max-transfers   <num>            maximum number of simultaneous uploads
                                        0=unlimited (default)
  -o, --overwrite           overwrite existing package even if size matches
  -k, --keep-files                   do not delete client data dir on uninstall
  -t, --temp-dir    <path>         tempory directory for package install
  --new-product-id  <product-id>
                           set an new product id when extracting opsi package
  --interface       <type>                 type of user interface
                      text                     text based interface
                      snack                    newt interface (default)
  -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
```

3.5. Tool: opsi V3 opsi-admin

New in opsi V3.

3.5.1. 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.

'opsi-admin' provides an interactive mode and a non interactive mode for batch processing from within scripts.

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

```
# opsi-admin -h

Usage: opsi-admin [-u -p -a -d -l -f -i -c -s] [command] [args...]

  -h, --help            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
  -l, --loglevel        Set log level (default: 2)
                        0=nothing, 1=critical, 2=error, 3=warning, 4=notice,
5=info, 6=debug
  -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 user name 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 -i`** or **`opsi-admin -d -i -c`**) 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.

3.5.2. Typical use cases

3.5.2.1. Delete product

The method is 'deleteProduct <productId>'. The command line request for deleting the product 'softprod' is:

```
opsi-admin -d method deleteProduct "softprod"
```

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

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

3.5.2.3. Client delete

```
opsi-admin -d method deleteClient <clientname>
For example:
opsi-admin -d method deleteClient pxevm.uib.local
```

3.5.2.4. Client create

```
opsi-admin -d method createClient <clientname> <domain>
For example:
opsi-admin -d method createClient pxevm uib.local
```

3.5.2.5. Client boot image activate

```
opsi-admin -d method setBootimage <OS-Produkt> <clientname>
For example:
opsi-admin -d method setBootimage win2k pxevm
```

3.5.2.6. Attach client description

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

3.5.2.7. Set pcpatch password

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

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

3.5.3. List of methods

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>,
       <packetVersion>, licenseRequired=0, setupScript="", uninstallScript="",
       updateScript="", alwaysScript="", onceScript="", priority=10,
       description="", advice="", productClassNames=('localBoot')
```

Creates a new localBoot product (wInst-Product).

```
method createNetBootProduct <productId>, <name>, <productVersion>,
       <packetVersion>, 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>,
       <packetVersion>, 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>,
       <packetVersion>, 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>
```

(For future use) Supplies an available license key of the specified product or the product license key which is assigned to the client.

```
method getLicenseKeys_listOfHashes <productId>
```

(For future use) 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 = []
```

(For future use) 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=None
```

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

```
method getPossibleProductInstallationStatus_list
```

Supplies the list of all installation stati (installed, not installed,...).

```
method getPossibleRequirementTypes_list
```

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

```
method getProduct_hash <productId>
```

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

```
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 depot server in charge of the specified client.

```
method getServerIds_list
```

Supplies a list of the known opsi depot server.

```
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 boot image 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-preloginloader 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 (policyId and licenseKey are for future use).

```
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 'undefined' so LDAP policies are in charge for this client.

4. Activation of non free modules: opsiclientd, license management, VPN-support

Even opsi is open source, there are some components which are not free at the moment. At this time (13 July 2009) the following components of opsi are not free:

- license management
- opsiclientd (Vista / Windows 7)
- VPN Support (not completely implemented yet)

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.

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.

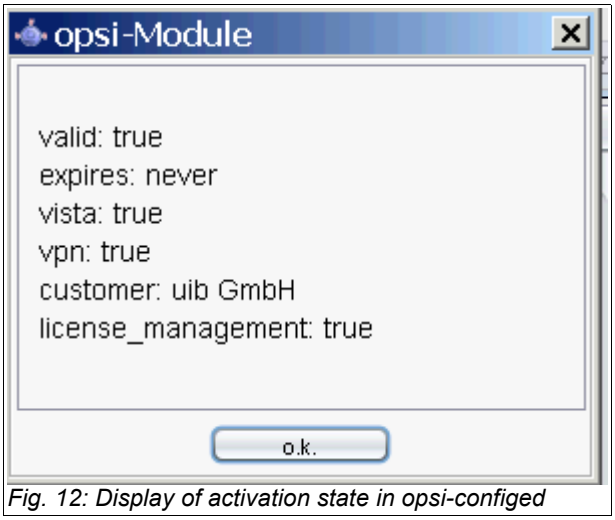


Fig. 12: Display of activation state in opsi-configed

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

4. Activation of non free modules: opsiclientd, license management, VPN-support

```
opsi-admin -d method getOpsiInformation_hash
{
"opsiVersion" : "3.4.0.0",
"modules" :
          {
          "customer" : "uib GmbH",
          "vista" : false,
          "license_management" : true,
          "expires" : "never",
          "valid" : true,
          "signature" : "THIS-IS-NOT-A-VALID-SIGNATURE",
          "vpn" : true
          }
}
```

5. preloginloader 3.4

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 by two software components:

On the client side at boot time before the user logs in the opsi preLoginLoader examines whether an update has to be installed for this client.

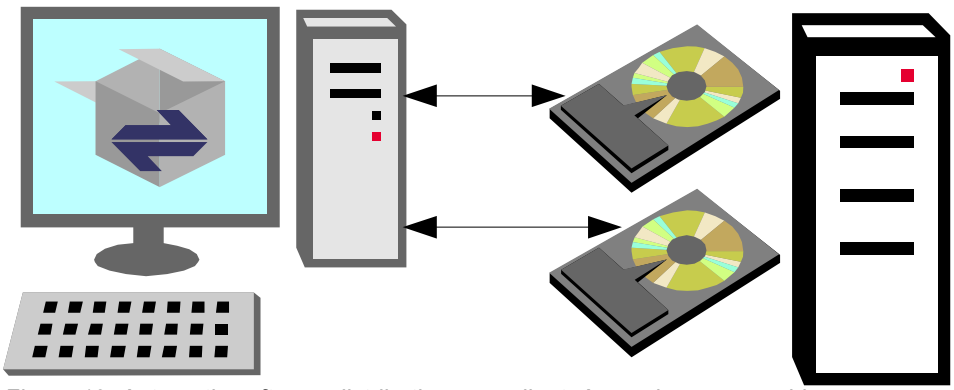


Figure 13: Automatic software distribution on a client. An opsi server provides configuration information and installable software packets.

If there are software packets to be installed on the client, the script processing program 'wInst' is being started to do the installation job. The server provides all the installation scripts and software packets 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 software packets can be installed with the 'wInst' program, they have to be prepared as opsi packets. For details see Chapter 'Integration of new software packets into the opsi software deployment'.

5.2. Two modes: opsiclientd and prelogin

Due to the major changes between Windows XP and Vista, the opsi preloginloader (with the exception opsi-winst) has been completely new implemented. The major component of this new implementation is the opsiclientd which replaces the old parts prelogin.exe and pcpatch.exe. Inside the opsi-preloginloader 3.4 packages there are both implementations which can be installed as different modes: prelogin or opsiclientd.

The opsiclientd is not free at the moment and subject to a co-funding project. In order to use the opsiclientd you need a activation entry for 'vista' in the activation file. For Details see Chapter 4 Activation of non free modules: opsiclientd, license management, VPN-support on page 39.

Wich mode will be installed is controlled by the product property 'client_servicetype' and the file /opt/pcbin/install/preloginloader/files/opsi/cfg/config.ini with the entry:

```
[installation]
;client_servicetype=prelogin
client_servicetype=opsiclientd
```

To check which is the default value for 'client_servicetype' at your server, call:

```
opsi-admin -d method getProductProperties_hash preloginloader
```

To set the default value for 'client_servicetype' at your server to 'prelogin' call:

```
opsi-admin -d method setProductProperty preloginloader "client_servicetype" "prelogin"
```

If you don't have activation for 'vista', you should use the 'prelogin' mode because the opsiclientd will not work without activation.

5.3. The new mode: opsiclientd

This new implementation has been done in the Python language which is also used for the server parts of opsi. For the installation are files are used, which are compiled by the py2exe program. This makes the installation independent from any existing python installation at the client.

The important enhancements 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 startup any more.

•Control via web service:
This interface is used for maintenance purpose at the moment. It will be used for central (time) controlled installations in future.

•Remote configuration:
The configuration data for the clients may be changed (globally or client specific) at the server by editing the 'general config' parameters

The opsi-preloginloader 3.4 consists of multiple components :

•opsiclientd: the central service

•notifier: information and communication window

•opsi-loginblocker: block the login until the installation has finished

The legacy prelogin.exe based service may be installed as well, but it is deprecated and should be replaced by opsiclientd based installations.

5. preloginloader 3.4

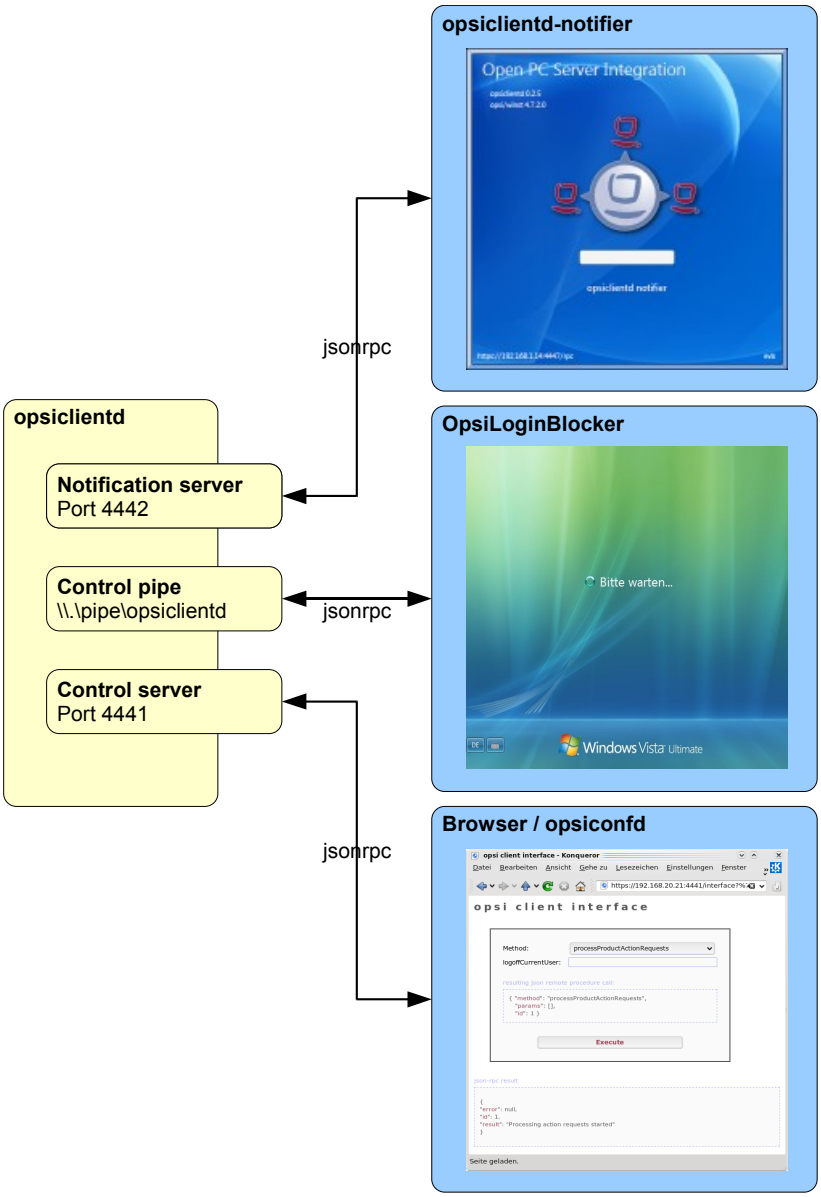


Figure 1: Scheme of the opsiclientd components

5.3.1. Installation

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

For a subsequent installation on a existing Windows system or for repair purposes there two possibilities that described more detailed in the opsi manual:

•Login with a administrative account, mount the opt_pcbin share of the opsi server and call the script **`install\preloginloader\service_setup.cmd`**

•Use the server side script opsi-deploy-preloginloader

At clients which are still integrated in opsi, the new preloginloader can be install with the standard opsi process by switching the required action to setup. At Vista clients with the former preloginvista installed, this product will be switched to not_installed. The product preloginvista may (and should) deleted from the server. The OS-Installation packages (winvista, winxppro, ...) must be updated too in order to work with the new preloginloader.

The preloginloader got a new product property 'client_servicetype'. The default is 'opsiclientd' which is the new one. In special cases you may switch to the legacy 'prelogin'. But this one contains no new extensions and is deprecated.

For deinstallation of the preloginloader the action request may be switched to 'uninstall'

5.3.2. opsiclientd

Core component of the preloginloader is the service opsiclientd. This service starts at the boot time.

The opsiclientd has the following tasks:

- Getting active if the configuration event takes place. The default event is 'gui_startup' which will fire (like the legacy version) at boot time and before login.
- Via web service (JSON-RPC) the opsiclientd contacts the opsi server and ask for configuration data and required actions.
- 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, mounting the depot share and update and start of the opsi winst 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.

5.3.3.1. opsiclientd_event_notifier

The *event_notifier* gets active if a event fires and for this event is configured that **`warning_time`** is > 0 (default = 0). In this case the user will see **`warning_time`** seconds a message windows with the in **`message`** configured text and a '*Start now*' button. Is **`user_cancelable`** = true, so will be also a '*Abort*' button enabled. When the **`warning_time`** is expired or if the user choosed '*Start now*' the actions and the *action_notifier* will be started.

At the default event **`gui_startup`**, the *event_notifier* is disabled by default. It is enabled and more important at events like **`vpn_startup`**.

Figure 2: opsiclientd event notifier

5.3.3.2. opsiclientd_action_notifier

The *action_notifier* presents the action progress and gives (if so configured) the possibility to cancel the process.

5. preloginloader 3.4

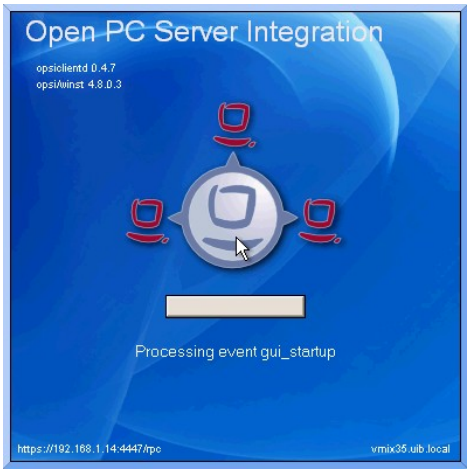


Figure 3: opsiclientd action notifier

5.3.4. opsi-loginblocker

The opsi login blocker at Vista is implemented as '**`credential provider filter`**'. It blocks all 'credential providers' until the release by the opsiclientd or the timeout.

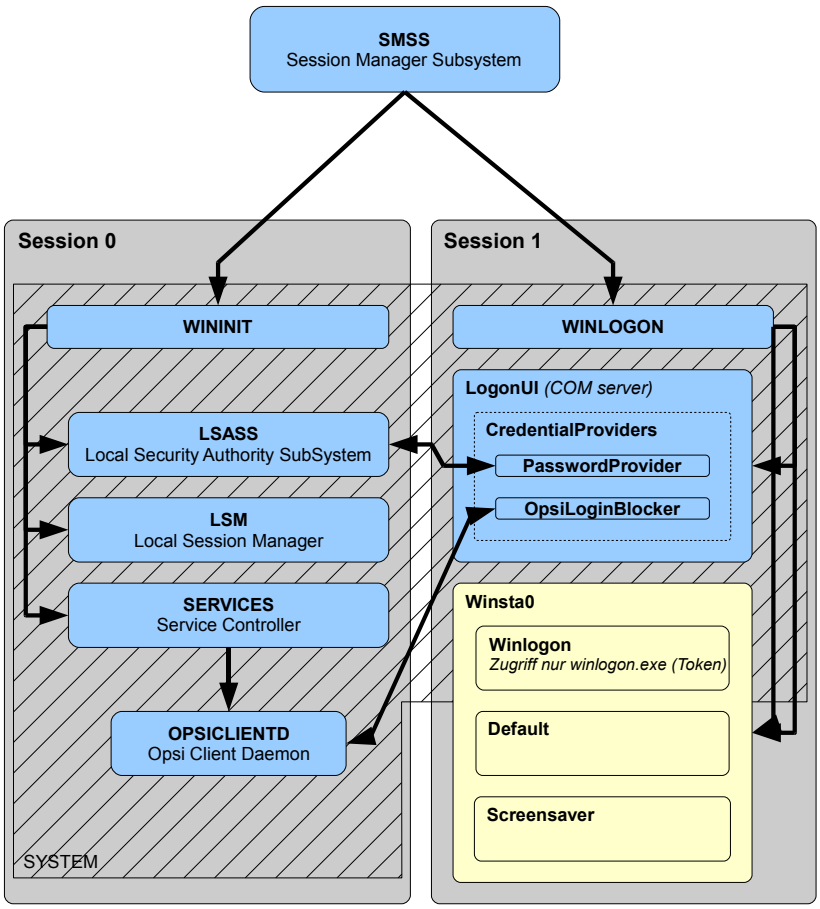


Figure 4: Scheme of opsiclientd an login blocker in Vista

The opsi login blocker at Win2K / Winxp is implemented as '**pgina**'. It blocks the **msgina.dll** until the release by the opsiclientd or the timeout.

5.3.5. Configuration

5.3.5.1. Configuration via configuration file

The configuration file is:

c:\program files\opsi.org\preloginloader\opsiclientd\opsicliend.conf

The configuration written in this file may be changed by different configuration data, which comes via web service after a successful connection to the opsi-server.

A sample 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: critical, 2: errors, 3: warnings, 4: notices
# 5: infos, 6: debug messages, 7: more debug messages, 9: passwords
log_level = 4

# Opsi host key.
opsi_host_key =

# 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

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;     config service settings                                                  -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[config_service]
```

5. preloginloader 3.4

```
# Service url.
# http(s)://<opsi config server address>:<port>/rpc
url = https://opsi.uib.local:4447/rpc

# Conection timeout.
connection_timeout = 10

# The time in seconds after which the user can cancel the connection
establishment
user_cancellable_after = 0

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     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 = %system.program_files_dir
%\\opsi.org\\preloginloader\\opsiclientd\\opsiclientd.pem

# The location of the server private key
ssl_server_key_file = %system.program_files_dir
%\\opsi.org\\preloginloader\\opsiclientd\\opsiclientd.pem

# The location of the static files
static_dir = %system.program_files_dir
%\\opsi.org\\preloginloader\\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 port where opsiclientd will listen for notification clients.
port = 4442

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

# Notifier application command
```

```
command = %system.program_files_dir%\\opsi.org\\preloginloader\\notifier.exe
-p %notification_server.port%

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

# RPC tool command
command = %system.program_files_dir
%\\opsi.org\\preloginloader\\opsiclientd_rpc.exe "%global.host_id%"
"%global.opsi_host_key%" "%control_server.port%"

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     action processor settings                                         -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[action_processor]
# Locations of action processor
local_dir = %system.program_files_dir%\\opsi.org\\preloginloader\\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 "https://%config_service.host%:%config_service.port%" /clientid
%global.host_id% /username %global.host_id% /password %global.opsi_host_key%

; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; -     events                                                            -
; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[event_daemon_startup]
type = daemon startup
active = false

[event_daemon_shutdown]
type = daemon shutdown
active = false

[event_gui_startup]
type = gui startup
message = Starting to process product actions. Attention: the computer may
restart. Please save all unsaved data now.
user_cancelable = false
block_login = true
lock_workstation = false
logoff_current_user = false
get_config_from_service = true
update_config_file = true
write_log_to_service = true
update_action_processor = true
event_notifier_command = %opsiclientd_notifier.command% -s notifier\\event.ini
event_notifier_desktop = current
action_notifier_command = %opsiclientd_notifier.command% -s
notifier\\action.ini
action_notifier_desktop = current
action_processor_command = %action_processor.command%
action_processor_desktop = current
```

```
[event_vpn_startup]
type = custom
active = false
wql = SELECT * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance
ISA 'Win32_NetworkAdapter' AND TargetInstance.Name = "TAP-Win32 Adapter V9"
AND TargetInstance.NetConnectionStatus = 2
message = Opsi will start software and hardware inventory on this computer.
You can continue your work in the meantime.
get_config_from_service = true
update_config_file = true
write_log_to_service = true
warning_time = 20
service_options = { "actionProcessingFilter": { "productIds": ["hwaudit",
"swaudit"] } }
event_notifier_command = %opsiclientd_notifier.command% -s notifier\\event.ini
event_notifier_desktop = current
action_notifier_command = %opsiclientd_notifier.command% -s
notifier\\action.ini
action_notifier_desktop = current
action_processor_command = %action_processor.command% /service_options
"%event_vpn_startup.service_options%"
action_processor_desktop = current
```

The above mentioned timeouts have the following relations:

1. If a event fires, the *event_notifier* shows **warning_time** seconds a message and according to the value of **user_cancelable** a 'Abort' button. Is the **warning_time** = 0 (default) the *event_notifier* don't starts.

2. After the **warning_time** the action starts, which means normally that the opsiclientd try to reach the opsi server using the **url** address.

3. If after **user_cancellable_after** seconds still no connection established, so the action_notifier will enable a 'Abort' button. Once the connection is established, there is no more possibility to abort.

4. If there no connection could be established in **connection_timeout** seconds, the opsiclientd abort the actions. To avoid a user from aborting, set **user_cancellable_after** = **connection_timeout** .

5.3.5.2. Configuration via web service (general config)

The opsiclientd configuration can be changed by the 'general config' at the server.

The entries in the general config have to been according to the following patterns: opsiclientd.<name of the section>.<name of the key>

Example:
opsiclientd.global.log_level = 4
set in the configuration file opsiclientd.conf in the section [global] the value of log_level to the value 4.

The following Figure shows how to change the server wide general configure via opsi configed

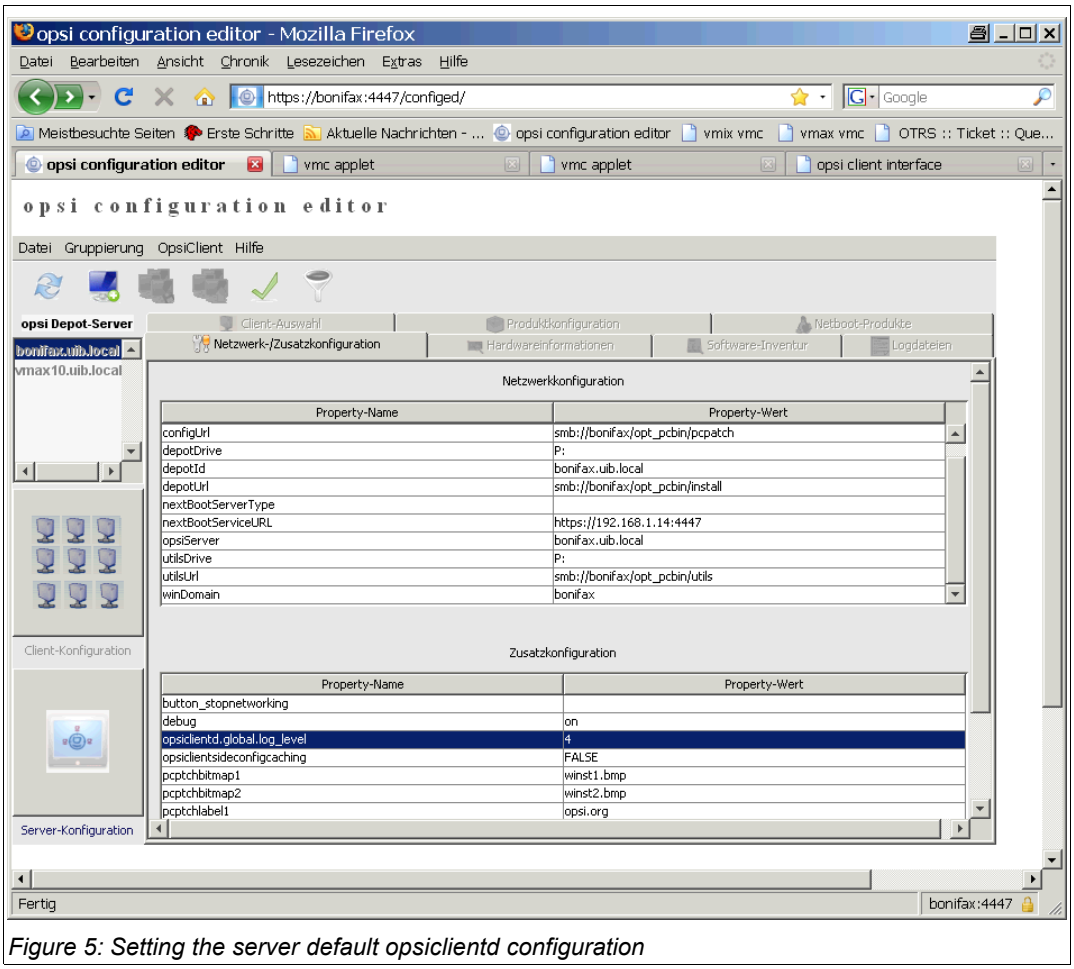


Figure 5: Setting the server default opsiclientd configuration

At the moment it isn't possible to manipulate these entries client specific via opsi configed (we working on this). So any client specific change at the general config must be done manually direct in the backend.

Here a example for the File31-Backend:

Excerpt from a <pcname>.ini file:

```
[generalconfig]
opsiclientd.global.log_level=6
```

Here a example for the LDAP-Backend (with JXplorer as LDAP-Browser):

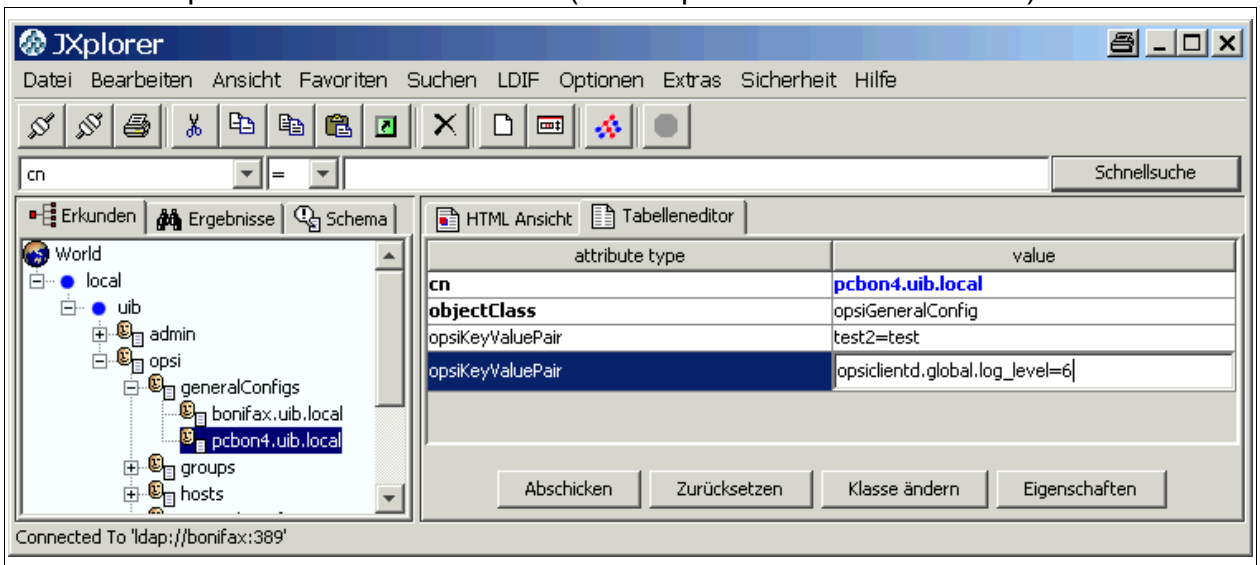


Figure 6: Client specific configuration of the opsiclientd at the LDAP-Backend using JXplorer

5.3.5.3. Configuration of different events

The following events are predefined:

•event_gui_startup
Default event at client boot - before login

•event_vpn_startup
Example of a 'custom event' where the fire condition is defined via a WMI-WQL query. In this case the condition is the activating of the VPN network interface:
wql = SELECT * FROM __InstanceModificationEvent WITHIN 2 WHERE TargetInstance ISA 'Win32_NetworkAdapter' AND TargetInstance.Name = "TAP-Win32 Adapter V9" AND TargetInstance.NetConnectionStatus = 2
where **"TAP-Win32 Adapter V9"** is the of the VPN network adapter which is specific for the used VPN software.

•event_daemon_startup
not implemented yet

•event_daemon_shutdown
not implemented yet

Setting the entry 'active = false' disables the event.

5.3.6. Logging

The opsiclientd logs to:

c:\tmp\opsicliend.log

All log informations will be transfered to the opsi server via web service. At the server you find these log infos at /var/log/opsi/clientconnect/<pcname>.log. They are presented in the opsi configed at the tab 'logfiles / client connect'.

Every line at the log has the pattern:
[<log level>] [<time stamp>] [message source] message.

There are the following log levels:

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

Example:

```
[4] [Feb 02 17:30:11] [opsiclientd]  Config read  (opsiclientd.pyo|1602)
[0] [Feb 02 17:30:11] [opsiclientd]  Opsiclientd version: 0.4.4.4  (opsiclientd.pyo|1816)
```

5. preloginloader 3.4

```
[0] [Feb 02 17:30:11] [opsiclientd]  Commandline:
C:\Programme\opsi.org\preloginloader\opsiclientd.exe  (opsiclientd.pyo|1817)
[0] [Feb 02 17:30:11] [opsiclientd]  Working directory: C:\WINDOWS\system32  (opsiclientd.pyo|
1818)
[4] [Feb 02 17:30:11] [opsiclientd]  Using host id 'vmix35.uib.local'  (opsiclientd.pyo|1819)
[4] [Feb 02 17:30:11] [opsiclientd]  Starting control pipe  (opsiclientd.pyo|1825)
[4] [Feb 02 17:30:11] [opsiclientd]  Control pipe started  (opsiclientd.pyo|1829)
[4] [Feb 02 17:30:11] [opsiclientd]  Starting control server  (opsiclientd.pyo|1834)
[4] [Feb 02 17:30:11] [opsiclientd]  Control server started  (opsiclientd.pyo|1843)
[4] [Feb 02 17:30:11] [opsiclientd]  Starting notification server  (opsiclientd.pyo|1848)
[4] [Feb 02 17:30:11] [opsiclientd]  Notification server started  (opsiclientd.pyo|1863)
[4] [Feb 02 17:30:11] [opsiclientd]  Event 'daemon_shutdown' is deactivated  (opsiclientd.pyo|
1770)
[4] [Feb 02 17:30:11] [opsiclientd]  Event 'net_startup' is deactivated  (opsiclientd.pyo|1770)
[4] [Feb 02 17:30:11] [opsiclientd]  Event 'daemon_startup' is deactivated  (opsiclientd.pyo|
1770)
[4] [Feb 02 17:30:12] [control server]  Control server is accepting HTTPS requests on port 4441
(opsiclientd.pyo|1164)
[4] [Feb 02 17:30:12] [control server]  Control server exiting  (opsiclientd.pyo|1170)
[4] [Feb 02 17:30:12] [opsiclientd]  gui startup event 'gui_startup' created  (opsiclientd.pyo|
1784)
[4] [Feb 02 17:30:12] [opsiclientd]  Waiting for gui startup (timeout: 120 seconds)
(opsiclientd.pyo|1872)
[4] [Feb 02 17:30:13] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:15] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:17] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:19] [event gui_startup]  Firing event '<event: gui_startup>'  (opsiclientd.pyo|
258)
[4] [Feb 02 17:30:19] [opsiclientd]  Processing event <event: gui_startup>  (opsiclientd.pyo|
1936)
[4] [Feb 02 17:30:19] [event wait_for_gui]  Firing event '<event: wait_for_gui>'
(opsiclientd.pyo|258)
[4] [Feb 02 17:30:19] [opsiclientd]  Executing:
C:\Programme\\opsi.org\\preloginloader\\opsiclientd_rpc.exe "vmix35.uib.local" "*** confidential
***" "4441" "setCurrentActiveDesktopName(System.getActiveDesktopName())"  (Windows.pyo|628)
[4] [Feb 02 17:30:19] [opsiclientd]  Gui started  (opsiclientd.pyo|1874)
[4] [Feb 02 17:30:19] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:21] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:21] [control server]  Authorization request from vmix35.uib.local@127.0.0.1
(opsiclientd.pyo|888)
[4] [Feb 02 17:30:21] [control server]  Authorization request from vmix35.uib.local@127.0.0.1
(opsiclientd.pyo|888)
[4] [Feb 02 17:30:21] [opsiclientd]  rpc setCurrentActiveDesktopName: current active desktop name
set to 'Winlogon'  (opsiclientd.pyo|2152)
[4] [Feb 02 17:30:22] [opsiclientd]  Process ended: 1736  (Windows.pyo|636)
[4] [Feb 02 17:30:22] [event processing]  Starting notifier application in session '0' on desktop
'Winlogon'  (opsiclientd.pyo|1295)
[4] [Feb 02 17:30:22] [event processing]  Executing:
C:\Programme\\opsi.org\\preloginloader\\notifier.exe -p 4442 -s notifier\\action.ini
(Windows.pyo|628)
[4] [Feb 02 17:30:23] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:25] [opsiclientd]  Getting config from service  (opsiclientd.pyo|1647)
[4] [Feb 02 17:30:25] [service connection]  Connecting to config server
'https://192.168.1.14:4447/rpc' #1  (opsiclientd.pyo|1235)
[4] [Feb 02 17:30:25] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:26] [service connection]  Connected to config server
'https://192.168.1.14:4447/rpc'  (opsiclientd.pyo|1247)
[4] [Feb 02 17:30:27] [opsiclientd]  Got config from service  (opsiclientd.pyo|1664)
[4] [Feb 02 17:30:27] [opsiclientd]  Trying to write config to file:
'C:\Programme\opsi.org\preloginloader\opsiclientd\opsiclientd.conf'  (opsiclientd.pyo|1607)
[4] [Feb 02 17:30:27] [opsiclientd]  No need to write config file
'C:\Programme\opsi.org\preloginloader\opsiclientd\opsiclientd.conf', config file is up to date
(opsiclientd.pyo|1637)
```

```
[4] [Feb 02 17:30:27] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:28] [opsiclientd]  Got product action requests from configservice
(opsiclientd.pyo|2294)
[4] [Feb 02 17:30:28] [opsiclientd]  No product action requests set  (opsiclientd.pyo|2302)
[4] [Feb 02 17:30:29] [opsiclientd]  rpc getBlockLogin: blockLogin is 'True'  (opsiclientd.pyo|
2065)
[4] [Feb 02 17:30:31] [opsiclientd]  Writing log to service  (opsiclientd.pyo|1675)
```

The opsi login blocker logging to to the Windows event log. If the log level is 8 and up there is also a log file: c:\tmp\opsi_loginblocker.log.

5.3.7. control server

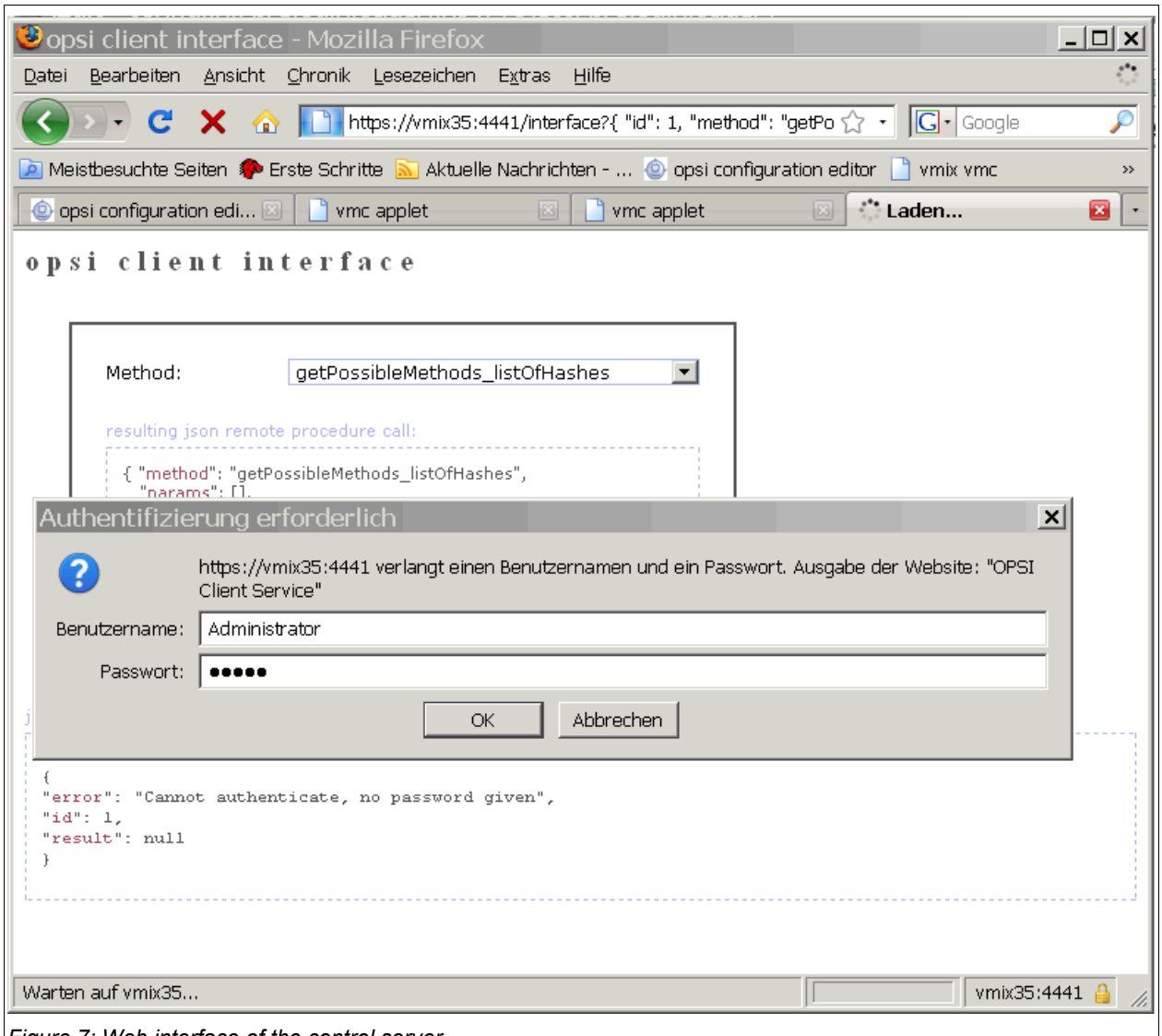


Figure 7: Web interface of the control server

The control server port may be used for a remote control of the opsiclientd. For security reasons we need an authentication. This authentication may be the local Administrator

with password (empty passwords are not allowed) or the full qualified client name withe client key (etc/opsi/pckeys) as password. At the moment the control server is used for maintenance purposes only.

5.3.8. Push Installation: opsi-fire-event.py

Example:

```
/usr/share/opsi/opsi-fire-event.py <client-id> gui_startup
```

5.4. The old mode: prelogin

The opsi preLoginLoader consists of four components: prelogin.exe, pcptch.exe, wInst32.exe and the optional Loginblocker. The prelogin.exe starts as a system service at boot time. That means the task prelogin.exe starts before the user login is available.

The main task of the prelogin.exe is to start the task pcptch.exe and grant access on the graphical user interface (otherwise the installation process wouldn't be displayed on screen). Pcptch.exe is started with the privileges of the local 'pcpatch' account (administrative account), which has been installed during PreLoginLoader installation. The program pcptch.exe mounts the network file shares which provide the software installation packets (and the PC configuration files if file backend is applied). The configuration for the installation process is provided by the opsi depot server. A description of this configuration information can be found in the related chapters at the end of this manual.

Then the pcptch.exe starts the opsi Installation program wInst.exe and passes the information which packets to install. After wInst.exe completed the installations, the status information is passed back to the opsi depot server. While the installation is still in progress, the loginblocker prevents the user login. When the installation is done, wInst.exe and pcptch.exe terminate and the user login screen is enabled.

Often an installation packet requires one or several reboots. In that case the installation script launches an immediate system restart. Then at system restart the installation process resumes control again and continues with the installation.

5.5. Blocking the user login with the opsi-Loginblocker

To prevent a user login before all installations completed, opsi provides the optional Loginblocker module.

5.5.1. opsi loginblocker under Windows 2000 to XP (prelogin and opsiclientd)

The Loginblocker is implemented as a gina.dll. Gina means „Graphical Identification and Authentication“ and is the official Microsoft hook to manipulate the login process. The opsi gina is a pgina.dll based on the project http://pgina.xpasystems.com.

If you already have a gina.dll installed which is different from the original msgina (e.g. Novells nwgina), you should not install the opsi-Loginblocker without consulting uib. It is possible to chain different gina.dll's, but therefore the installation has to be customized. Proper chaining of Gina DLLs is a quite critical task and might result in a locked up computer if done improperly.

Whether the Loginblocker is installed or not this is configured by the switch LoginBlockerStart=on/off in section [preloginloader-install] of the client configuration.

5.5.2. opsi loginblocker under Vista & Co (only opsiclientd)

The opsi login blocker at Vista is implemented as '**`credential provider filter`**'. It blocks all 'credential providers' until the release by the opsiclientd or the timeout.

5.6. Subsequent installation of the opsi-preloginloaders

The information about the 'Subsequent installation of the opsi-preloginloaders' you will find in the opsi-getting-started manual (Chapter 'First Steps').

6. Localboot products: automatic software distribution with opsi

6.1. opsi standard products

6.1.1. opsi-preloginloader

The 'opsi-preloginloader' packet contains the installation and update mechanism of the opsi-preloginloader.

6.1.2. opsi-wlnst

The 'opsi-wlnst' packet is a special case. It includes the actual 'opsi-wlnst' winst.exe, which is updated by the preloginloader packet itself. The preloginloader checks the server for an update of winst.exe and then copies the new winst.exe (and it's files) to the client.

6.1.3. Javavm: Java Runtime Environment

The product 'javavm' installs the required Java 1.6 runtime environment (required for opsi-configed) on the clients.

6.1.4. opsi-admin

The product 'opsi-admin' offers some utilities and a local installation of the opsi-configed.

6.1.5. Swaudit and hwaudit: Products for hardware and software inventories

The products hwaudit and swaudit provide the hardware and software inventories.

The hardware data are acquired using WMI and written to the hardware inventory via opsi-webservice.

The data for the software inventory are taken from the registry (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall) and passed to the inventory server via opsi-webservice.

Both products are based on python and require the python language package installed.

6.1.6. opsi-template

Template for you own opsi scripts.

You may extract this template with:

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

it is also possible to rename it at the same time:

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

6.1.7. python

It is the python runtime environment for python based products like swaudit and hwaudit.

6.1.8. xpconfig

Package for customizing the GUI and Explorer settings (not only) for XP.

6.2. Integration of new software packets into the opsi software deployment.

The information about the 'Integration of new software packets into the opsi software deployment.' you will find in the opsi-getting-started manual.

7. Netboot products: Automated OS installation and more

7.1. Unattended automated OS installation

7.1.1. Overview

Steps of a re-installation:

- Using PXE-Boot:
  - Choose the client which has to be installed with the utility opsi-configed or opsi-admin.
- At the next reboot, the client detects (via PXE-Bootprom) the re-installation request and loads the boot image from the opsi depot server.
- Using CD-Boot:
  - The client boots the boot image from the opsi-bootcd.
- The boot image starts and asks for confirmation to proceed with the re-installation. This is the only interactive question. After confirming this, the installation proceeds without any further request for interaction.
- The boot image partitions and formats the hard disk.
- The boot image copies the required installation files and configuration information from the depot server to the client and initiates a reboot.
- After the reboot the client installs the OS according to the provided configuration information without any interaction.
- Next the opsi PreLoginLoader is installed as the opsi installer for automated software distribution.

- The automated software distribution then installs all the software packages as defined in the client's configuration.

7.1.2. Preconditions

First of all an opsi depot server has to be installed.

The client PC has to be equipped with a bootable network controller. Most recent network controllers provide this functionality (PXE boot), also recent network controllers which are integrated on the PC's main board. The PXE software, which is stored in the 'bootprom' of the network controller, controls the boot process via network according to the BIOS boot device sequence. Usually the boot sequence has to be set in the BIOS, 'network-boot' has to be the first boot device.
On request also the support of 'bootp' bootproms is available.

The opsi installation package for the OS to be installed needs to be provided on the depot server. In the following we assume Windows 2000 to be the OS to install.

7.1.3. PC-client boots via the network

The PXE firmware gets activated at startup of the PC. Part of the PXE implementation is a DHCP client.

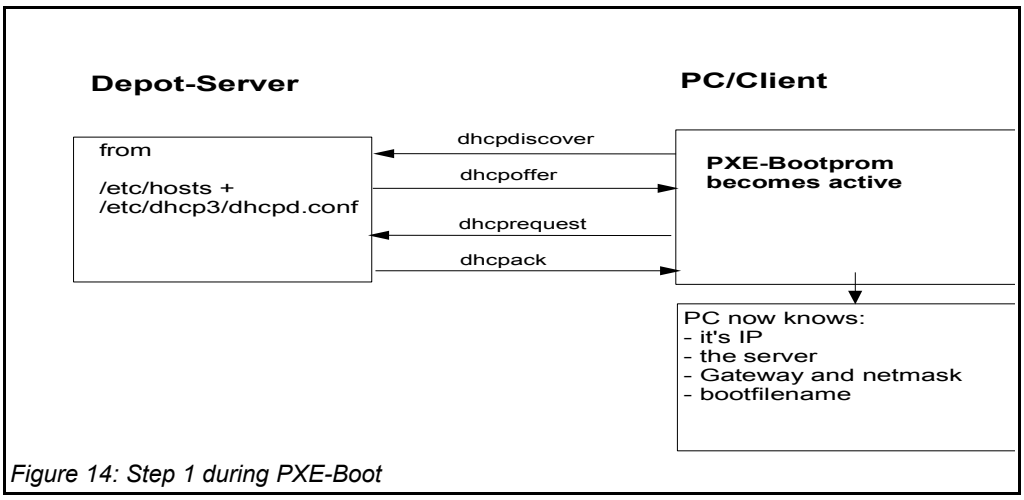


Figure 14: Step 1 during PXE-Boot

At first the PC only knows its hardware Ethernet address (MAC), consisting of six two-digit HEX characters.

The firmware initiates a **DHCPDISCOVER** broadcast: “I need an IP address, who is my DHCP-Server?“

The DHCP-Server offers an address (**DHCPOFFER).**

DHCPREQUEST is the response of the client to the server if the IP address is accepted. (This is not an obsolete step as there could be more than one server in the network.)

The server sends a **DHCPACK** to acknowledge the request. The information is sent to the client again.

You can watch this process on the display, for the PXE-BOOTPROM displays some firmware information and its 'CLIENT MAC ADDR'. The rotating pipe-symbol is displayed during the request. When an offer was made it is replaced by an '\' and you get the transmitted information (CLIENT IP, MASK, DHCP IP, GATEWAY IP).
A short while later you should get a response like this: 'My IP ADDRESS SEEMS TO BE .....'.

This process makes the PC a regular, fully configured member of the network.

The next step is to load the boot file (boot image) given in the configuration information.

7.1.3.1. Loading pxelinux

The boot image is loaded via trivial file transfer protocol (tftp). The displayed message is „**LOADING**“. tftp is a rather old and simple protocol to transfer files without authentication. In fact, all data available via tftp is available to everyone in the network. Therefore the tftp access is limited to one directory, which is usually '/tftpboot'. This directory is specified in inetd (internet daemon, /etc/inetd.conf), which will start the tftp daemon 'tftpd' if requested. The start command as noted in inetd.conf is something like **`tftpd -p -u tftp -s /tftpboot.`**

The PXE boot-process is multi-stage:

Stage 1 is to load and start the file submitted as part of the address discovery process (usually /tftpboot/linux/pxelinux.0).

The program 'pxelinux.0' then looks for configuration and boot information in '/tftpboot/linux/pxelinux.cfg'. It first looks for a PC specific file with a name based on the hardware ethernet address (MAC) of the network controller with a leading 01. The filename for the controller with the hardware ethernet address 00:0C:29:11:6B:D2 would be 01-00-0c-29-11-6b-d2. If the file is not found, 'pxelinux.0' will start to shorten the filename (starting at the end) to obtain a match. If this process ends without result, the file 'default' will be loaded. This file only contains the instruction to boot from the local hard disk. In this case the PC won't install anything and will just start the current OS from hard disk.

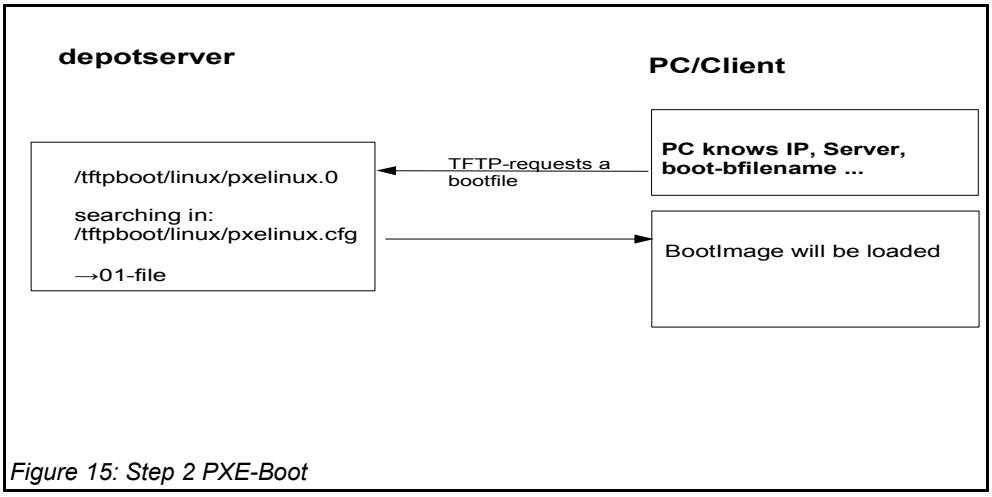


Figure 15: Step 2 PXE-Boot

To initiate the re-installation of a certain PC, a loadable file is prepared for the program 'pxelinux.0'. In order to do so, the opsi reInstallationsManager creates a PC custom file in '/tftpboot/linux/pxelinux.cfg'. Part of this file is the command to load the installation boot image. Also this file contains the client key to decrypt the pcpatch-password. This file is created as a 'named pipe' and therefore disappears after being read once. More details about this in the chapter on security of file shares.

Based on the information the 'pxelinux.0' got from the 'named pipe', the actual bootimage is loaded from the opsi depot server via tftp. The bootimage is based on a linux kernel (/tftpboot/linux/install) within an appropriate initrd file system (/tftpboot/linux/miniroot.gz) and has a size of approximately 40MB.

7.1.4. Boot from CD

Similar to the tftp boot via PXE-bootprom, the installation boot image can be booted from the opsi bootcd.

This might be recommended under the following conditions:

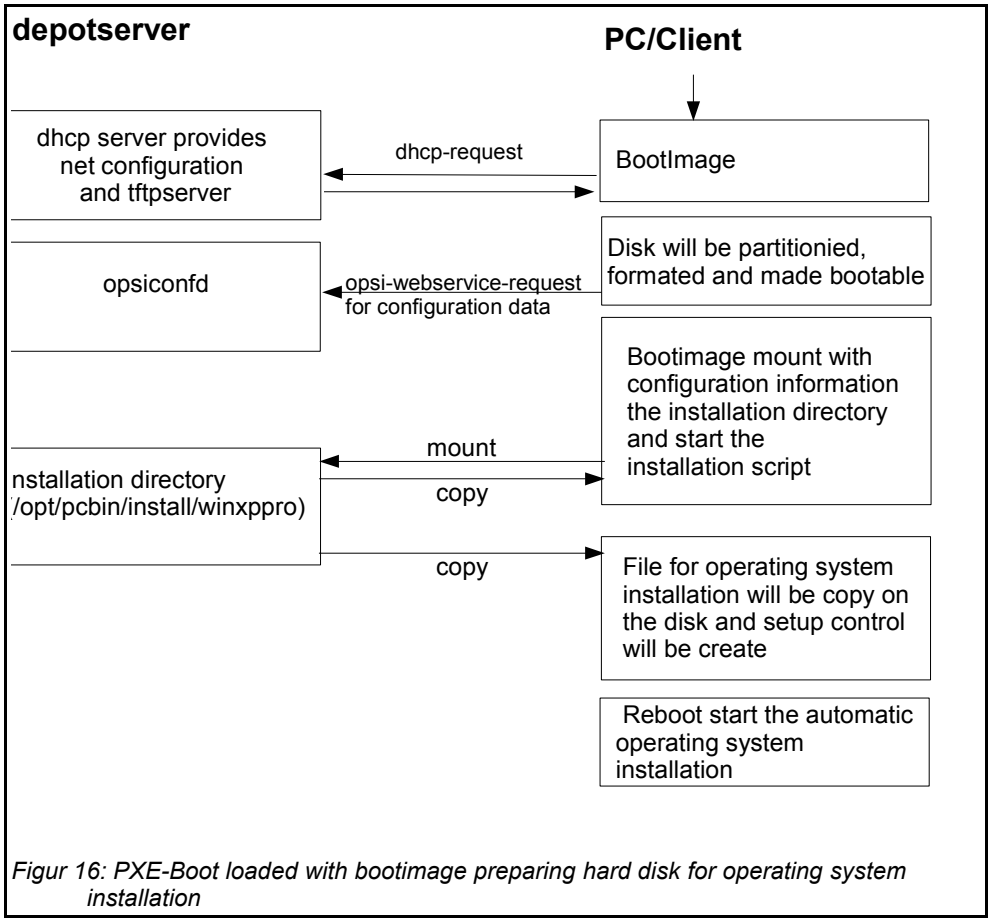
- the client has no PXE bootprom;
- there is no dhcp;
- there is a dhcp but it isn't allowed to configure any client data and the hardware addresses of the clients are unknown;
- there is a dhcp but it isn't configured for this demand.

According to different situations, several information has to be provided for the CD boot image by interactive input. The most simple case is to provide no further information. Eventually the clients hostname can be passed by hn=<hostname>. Using the option ASK_CONF=1 several parameters can be queried. Pressing F1 at the CD prompt shows the syntax.

7.1.5. The linux bootimage prepares for reinstallation

The bootimage again performs a dhcp request and configures the network interface according to the perceived information. Afterwards the configuration data for the client will be loaded via opsi-webservice.

The configuration data provides the information about the server in charge, the file share and the name of the installation script.



Figur 16: PXE-Boot loaded with bootimage preparing hard disk for operating system installation

It also holds the information on how to partition the hard disk, what file system to use and which operating system to install. Also it provides the encrypted password to connect the file share.

These informations will be combined with some information taken from the dhcp response and then be passed to the installation script for further processing.

Then the password for the user 'pcpatch' will be decrypted with the transferred key to mount the installation share and then call the installation script from the mounted share to start the installation of the operating system. What specific operations the script performs depends on the operating system which is to be installed. Below the steps of a Windows XP installation will be described.

Prepare the disc: On the hard disk the bootimage creates a new partition (of size 6 GB), formats it and installs a bootable ntloader kernel.

Copy the installation file: The files required for OS installation and the setup files for the opsi-PreLoginLoader (which is the opsi software distribution pack) will be copied from the server file share (e.g. /opt/pcbin/install/winxppro/i386) to the local hard disk.

Maintain the configuration informations: Some of the configuration and control files contain replacement characters, which will be patched before starting the actual installation. With a specified script (patcha-script) the placeholders will be replaced with parameters taken from the information packet, which is built from configuration files and the dhcp-response. For example the file 'unattend.txt', which is the control file for unattended OS Installation, will be patched with specific information like host IP, client IP, client name, workgroup, default gateway etc..

Prepare Reboot: Bootrecords will be installed which will start the Windows setup program at the next reboot. The patched 'unattend.txt' is passed to the setup as the control file for unattended installation.

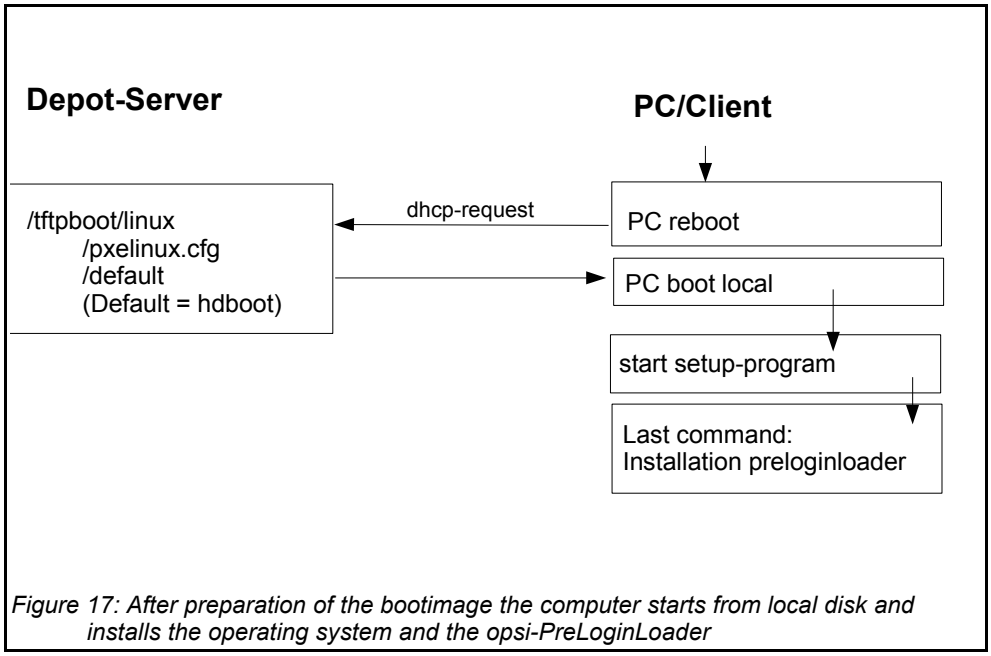


Figure 17: After preparation of the bootimage the computer starts from local disk and installs the operating system and the opsi-PreLoginLoader

Reboot: During the previous boot, the named pipe (which is indicating a request for installation) has been removed by reading it once. So the next PXE boot will load the

default netboot response, which executes the command 'hdboot'. The local boot loader will be started and the setup for operating system installation starts.

These steps are controlled by an OS specific python script (e.g. winxp.py for the Windows XP installation). The bootimage provides a python library (description in the opsi-bootimage handbook).

7.1.6. Installation of OS and opsi-preLoginLoader

The OS installation is based on the Microsoft unattended setup. Part of this is the standard hardware detection. In addition to the possibilities given during an installation from non-OEM or slipstreamed installation media, drivers and patches (i.e. service packs) can be installed during the initial installation, making the separate installation of drivers obsolete.

One feature of the unattended installation is the possibility to initiate additional installations after the main installation is finished. This mechanism is used to install the opsi preLoginLoader, which implements the automatized software distribution system. An entry in the registry marks the machine as being still in the 'reinstallation-mode'.

The final reboot leads to starting the opsi preLoginLoader service for software distribution prior to the first user login. Based on the value of the aforementioned registry key the opsi preLoginLoader switches into 'reinstallation-mode'. Therefore, regarding the configuration status of each software packet, each packet which is marked as action status **”setup”** or installation status **”installed”** within the configuration of that client will be installed. After all the designated client software has been installed, the reinstallation process is finished and the internal status is switched back from 'reinstallation-mode' to 'standard-mode'. In 'standard-mode' only software packages that are marked as action status **”setup”** will be installed.

7.1.7. How the patcha program works

As mentioned above the information collected from dhcp and opsi-webservice will be used to patch some configuration files as e.g. 'unattend.txt'. The program used for patching is the script '/user/local/bin/patcha'.

This script replaces patterns like **#@flagname(*)#** in a file with values taken as 'flagname=value' from a control file (default input is '/proc/cmdline'). In the files that have to be patched, the search and replace pattern must start with '#@', might have an optional '*' after the flagname and must have one or more trailing '#'.

So by calling 'patcha <filename>' the file '<filename>' will be patched with information taken from '/proc/cmdline'.

Calling 'patcha' without any parameters will show all the 'flagname=value' entries from '/proc/cmdline'.

A different input file ('another_cmdline') can be passed to 'patcha':

patcha -f another_cmdline

Without any other parameter 'patcha' will show the information taken from 'another_cmdline'. This input file must have 'cmdline'syntax, which means to be entries like '<flagname>=<value>' separated by space.

patcha -f another_cmdline patchfile

This will patch 'patchfile' with substitutions taken from 'another_cmdline'.

```
Version 0.93 23.10.2003 (c) J.W.

Usage:

$prog [-v] [-h] [-f cmdline] [file]
Options:    -v   show version only
            -f   another input file (cmdline)
            -h   this help
$prog patches files using Flag=value patterns from /proc/cmdline (default).
Without any parameters will show the values taken from /proc/cmdline.
```

Caveat: patch a patches only the first pattern of each line.
Each pattern will be expanded (or reduced) to the length of the value to be replaced with and then replaced. Trailing chars will not be affected.
Examples:

With the input file 'try.in'

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

and the file 'patch.me' to be patched:

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

the result will be:

```
./patcha -f try.in patch.me

cat patch.me
<very_long_substitution>
<t2>
<very_long_substitution>
<t2>
<very_long_substitution>
<t2>
<very_long_substitution>
<t2>
<very_long_substitution><#@tag1#####>
<t2><#@tag1#>
```

7.1.8. Structure of the unattended installation products

The information about the 'Structure of the unattended installation products' you will find in the opsi-getting-started manual.

7.1.9. Simplified driver integration with symlinks

The information about the 'Simplified driver integration with symlinks' you will find in the opsi-getting-started manual.

7.2. Ntfs image (write and restore)

The products 'ntfs-write-image' and 'ntfs-restore-image' are utilities to save and restore client partition images. Target (and source) for the image file has to be on the opsi depot server and will be transferred per ssh (user pcpatch) as specified as an product property.

7.3. memtest

The product 'memtest' is a utility to perform a client memory test.

7.4. hwinvent

This product delivers a hardware inventory of the client.

7.5. wipedisk

The product 'wipedisk' overwrites the complete hard disk (partion=0) or several partitions with different patterns. The number of consecutive write operations to perform is specified as the product property 'iterations' (1-25).

8. opsi license management

8.1. The opsi license management module - a co-financed opsi extension

8.1.1. Overview

The opsi license management module is designed for handling the software licenses for proprietary software installed on opsi clients.

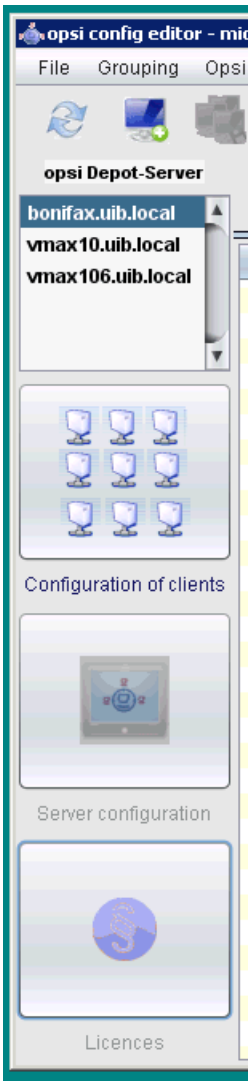


Figure 8: Start the license configuration

The main features are:

- ➔ Integrating the license management into the opsi config editor, which is the standard opsi user interface for managing the software configuration of opsi clients.
- ➔ Software license management features, which are insertion, reservation, (automated) assignment, release and deletion of license keys and license pools.
- ➔ Several types of licenses are available:
  - standard single licenses (a single license key assigned to a single client)
  - volume licenses (a single license key valid for a certain amount of installations)
  - campus license (a single license key valid for an unlimited amount of installations within the company/site)
  - client bound licenses (which is a single license valid for a dedicated client only, e.g. OEM licenses).
- ➔ Release of license assignment after deinstallation of the corresponding software.
- ➔ Manual editing of license data, e.g. for software which is not deployed by opsi.

- ➔ Reporting functions showing the opsi license assignment and the software installations detected by the software inventory module.

The license management module is a separate window integrated into the opsi config editor and can be started with the button „Licenses“, which is enabled if the license management module is unlocked within the opsi configuration („license management“ from the main menu /help/modules).

8.1.2. Acquisition and Installation

The opsi license management module is a co founded opsi extension module, which is available to the participants of the co funding project, who have payed a certain amount of the development costs (which is 1.000€ plus tax). The module will be available to the community when all the development costs have been funded.

The opsi license management module is part of the opsi release 3.4 and will be available if this feature is unlocked. For installation details see the installation manual.

8.2. License pools

For every type of license a *license pool* has to be defined. The license pools represent the use cases of licenses and provide the license keys for installing the licensed software on the clients.

When starting the license management module, the first tab holds the administration of license pools:

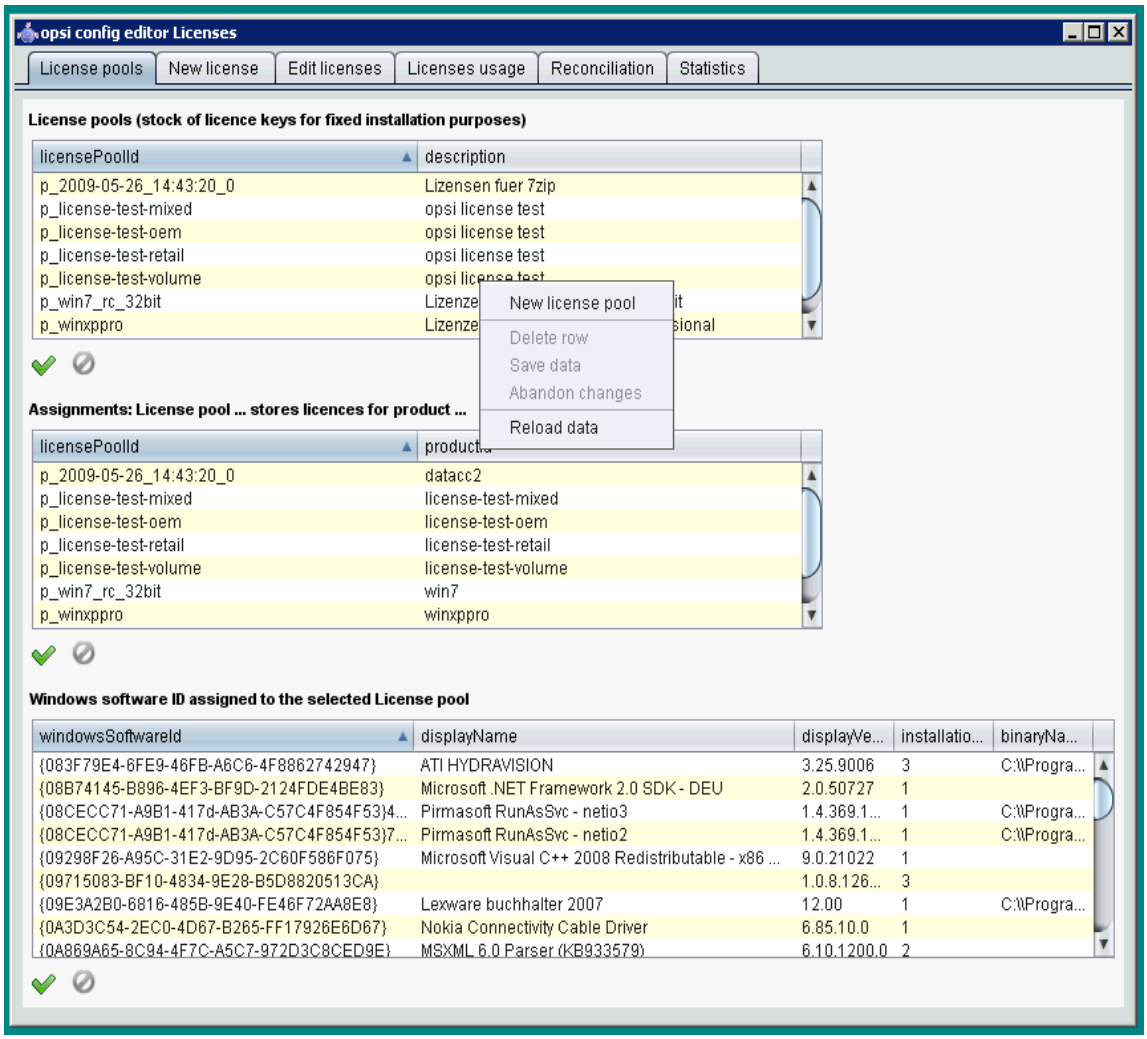


Figure 9: Tab "License pools" from the license management window

The upper part contains the table of available license pools. The context menu provides several functions for managing license pools, especially to insert a new license pool. When editing, the green check mark changes to red and the cancel option is enabled. By clicking the red check mark the changes can be saved, or canceled by clicking the cancel option (also available from the context menu).

8.2.1. License pools and opsi products

In most cases an opsi product installing licensed software (e.g. acrobat writer) is connected to a single license pool providing the license keys.

More complex are the license management requirements of an opsi product installing several software packets, which are subject to different license conditions. For instance an opsi packet "designer" installing *Adobe Photoshop* as well as *Acrobat Writer*. In this case the opsi product must retrieve license keys from different license pools.
Also there could be different opsi packets requesting licenses from the same pool. So there might be a complex and ambiguous relationship between opsi products and license pools.

The middle part of the license pool tab manages the relationship between license pools and *productIds (*opsi products).

As it is with all tables of the license management module, clicking on the column header will sort the table content by the column content. Clicking again inverts the order (ascending or descending).

Sorting can be used to display the connections between opsi products and license pools. Sorting by 'opsi product' displays all license pools connected to a certain product, whether sorting by 'license pool' shows which opsi products are connected to a certain license pool.

The context menu provides an option for inserting a new relationship from 'opsi product' to 'license pool'. An empty row is inserted on top of the table. Clicking into the field 'licensePoolId' or 'productId' displays a dropdown with available options.

8.2.2. License pools and software IDs

The third section of the 'License pools' tab holds the Windows software IDs connected to the currently selected license pool (in the first section of the tab).

A Windows software ID is a unique key identifying a software packet and will be written to the registry during installation. These software IDs are also used by the opsi software inventory module to identify which software is actually installed on the client.

The assignment of software IDs to the current license pool can be changed by setting or removing the selection (ctrl-click or shift-click). From the context menu the display can be toggled between showing all available software IDs detected by the software audit or just showing the software IDs connected to the current license pool.

Displaying the relationship between Software IDs and license pools is useful for comparing the number of actual software installations (detected by the software audit) with the number of legal installations available from the license pool (Tab 'Statistics', see chapter 8.7).

8.3. Setting up licenses

Setting up a license for being provided by a license pool requires several steps. The second tab 'New license' is for setting up and editing licenses.

On top is the table of available license pools to select the license pool the new license is assigned to.

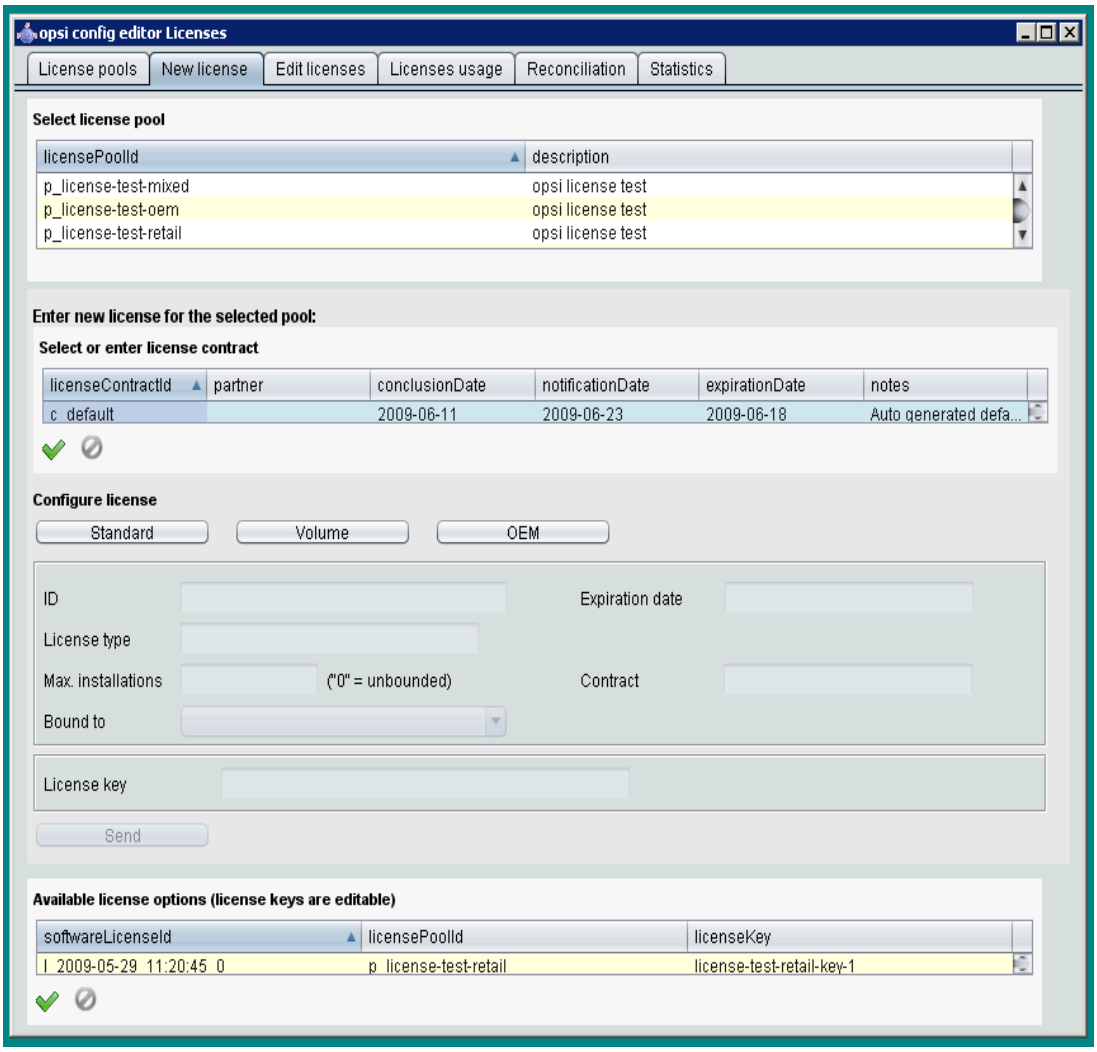


Figure 10: Tab "New license" from the licence management window

Before continuing with the next steps (chapter 8.3.2 ff.), some basic concepts and terms of license management have to be introduced:

8.3.1. Some aspects of the license concept

Licensing means the actual deployment of a permission to use a software by installing the software on a client. This might (but doesn't have to) include the use of a special **license key**.

The **software license** is the permission to install and use a software as defined by the *license contract*. Within the opsi database a *software license* is identified by a *softwareLicenseId*.

There are several types of software licenses (volume license, OEM license, time limited license etc.) which are the different **license models**. A software license is based on a **license contract**, which is defining and documenting the license regarding the juristic aspects.

A **license option** defines the option to use a *software license* for a selected *license pool*. Within opsi the *license option* is defined by a combination of *softwareLicenseId* and *licensePoolId*. This includes the actual *licenseKey* (if required).

Finally the **license usage** documents the use of a *license* by assigning the *license option* to a client. This is the legal and implemented *licensing* of a software defined by the combination of *softwareLicenseId, licensePoolId,* the unique client name *hostId* and (if required) the *licenseKey.*

8.3.2. Registering a license contract

After selecting the license pool for the new license option, the next step is to register the license contract the license is based on.

The section "Select or enter license contract" (from tab "New license") defaults to standard contract with ID *default.* The default setting can be used if the license contract is implied by purchasing the software or the contract is documented some other way.

Otherwise the contract can be selected from the table or a new contract can be registered from the context menu.

The license contract dataset comes with data fields for *partner*, *conclusion date*, *notification date* and *expiration date*. The entry field *notes* can hold some additional notes like the location where the contract document is kept.

The unique contract ID *(licenseContractId)* is for identifying the license contract in the license management database. When entering a new license contract, a new unique ID is constructed based on the current date and time stamp. This ID can be changed before saving the new data set. When saving the data, the opsi service checks whether the ID is unique. In case it is not, a new ID is generated and cannot be changed any more.

8.3.3. Configuring the license model

The third part of the Tab "New license" is named "Configure license" and is for registering the license model and license data.

Three types of license models are available:

- Standard
- Volume
- OEM

Each Option is represented by a button. Clicking a button, the form is filled with data for that type of license model.

License model **Standard** means that this license is valid for a single installation on an arbitrary client. So the license key (if any) is valid for a single installation only.

A **Volume** license is valid for a certain number of installations (given by "Max. installations"). In this case the optional license key is used for that number of installations. Setting "Max. installations" to "0" means, that the number of installations is unlimited within the same network (**campus license**).

In case of an **OEM** license the license is valid for a dedicated client only. Clients that come with a vendor pre installed operating system often have this type of license for the pre installed OS and software packets.

After clicking a button, the automatic data setting includes generating a unique ID (date and time stamp). This ID can be changed as desired.

It depends on the type of license model, which of the other fields can or cannot be changed.

The field "Expiration date" defines the expiration date of the license in a technical sense.

8.3.4. Saving the data

The "Send" button sends the data to the opsi service to save them permanently to the opsi data base (if they are consistent and no errors occur).
While proceeding this, data records will be generated for the new *software license* based on the selected *software contract* and the new *license option* assigned to that.

The list of available *license options* at the bottom of the window will be refreshed with the new *license option* selected. If necessary, the *license key* can be changed then.

8.4. Editing licenses

For ninety percent of the use cases editing the license data with the tabs "License pools" and "New license" will do. But there might be some special cases affording to edit license data more specific and explicit.

Therefore the tab "Edit licenses" presents the license data in three tables close to the internal data structure, allowing to adapt the data for some special cases.

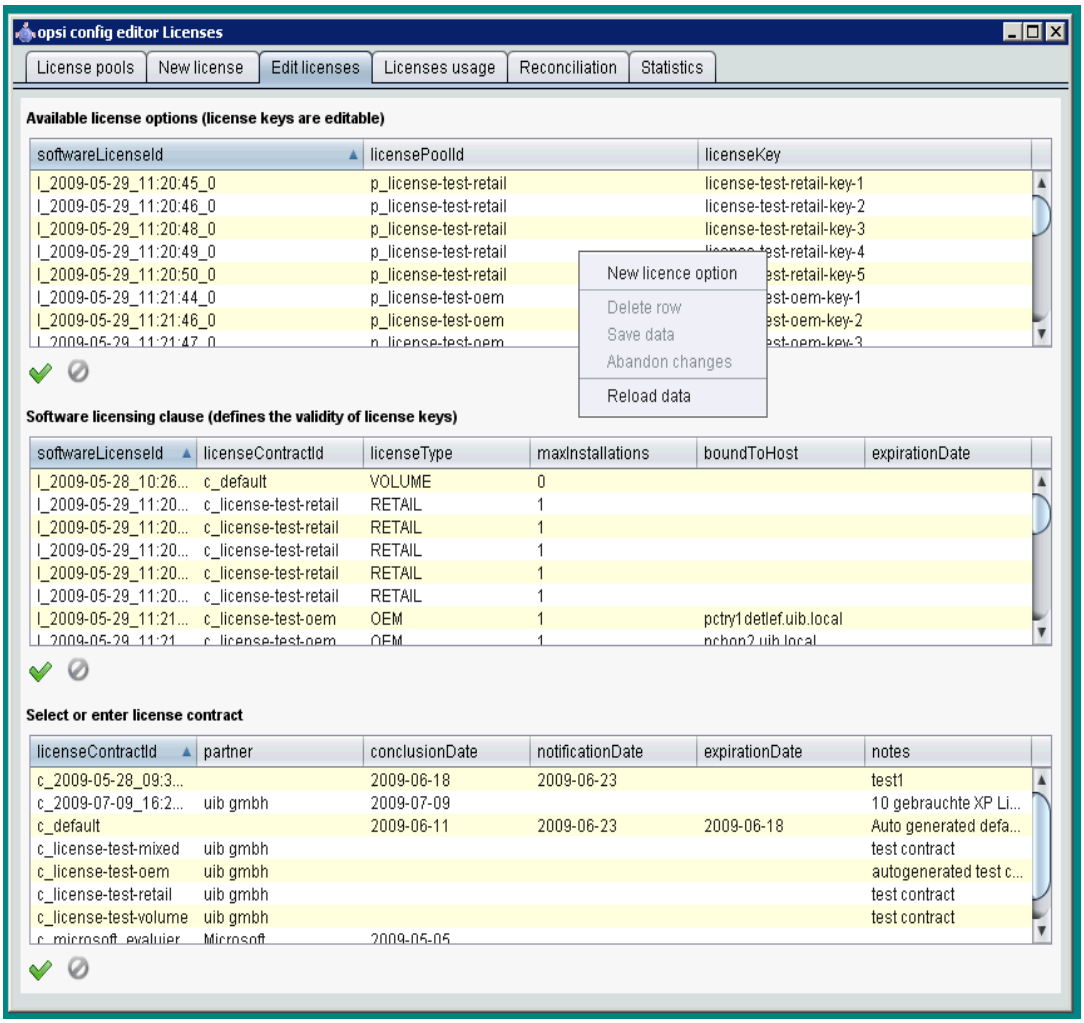


Figure 11: Tab "Edit licenses" from the license management window

Based on this direct data access, the following chapter shows how to configure a special license, like the Microsoft Vista or Windows 7 professional downgrade option for installing Windows XP.

8.4.1. Example downgrade option

downgrade option means, that instead of the software purchased, also the preceding version can be installed. For instance installing Windows XP based on a Windows Vista license. In this case, the license key also can be used for an installation, which it wasn't meant for in the first place.

In the opsi license model this case can be configured like this:

From the tab "New license" the Vista license is to be registered as usual, resulting in a new *license option,* displayed in the list of license options at the bottom of the window. This new license option is based on a new *software license* identified by *softwareLicenseId*.

This *softwareLicenseId* is needed for the further configuration steps. You can keep it in mind or copy it with drag&drop. You can as well look for the ID in the "Available license options" list of the "Edit licenses" tab. The context menu supports copying the ID.

The important step now is to connect this *softwareLicenseId* to an additional *license pool.*

Therefore a new record has to be registered from the "Available license options" table of the "Edit licenses" tab. The fields of the new record have to be filled with the *softwareLicenseId* and the ID of the additional *license pool* (in this case the pool for Windows XP licenses). For installing Windows XP based on this license, an applicable Windows XP *license key* already in use by another client has to be added.

After saving the new record, there are two different *license options* based on the same *software license*. The opsi service counts the use of either of them as an installation deducting from the maximum installation count. So in case of a downgrade license (with maxInstallations = 1), the opsi service delivers a *license key* for a Vista installation or for a XP installation, but not for both of them.

8.5. Assignment and release of licenses

Using a license option by installing the software on a client results in the actual licensing (which is the use of the license option).

In the opsi context installations are done script based and automatically, whereas the client running the Winst script invokes some calls to the central opsi service.
The following chapters introduce some of these service calls, which are relevant for the license management. For further information about Winst and opsi commands see the documentation on Winst and opsi.

8.5.1. opsi service calls for requesting and releasing a license

The opsi service call for requesting a license option and retrieving the license key for doing the installation (as transmitted by a Winst script) is

`getAndAssignSoftwareLicenseKey`

The parameters to be passed are the client hostID (hostID of the client where the software is to be installed) and the ID of the *license pool* the license is requested from. Instead of the *licensePoolId* also an *opsi product ID* or a *Windows Software ID* can be passed, if they are connected to a *license pool* within the opsi license management.

The use of a license option can be released by calling

`deleteSoftwareLicenseUsage`

Again the parameters to be passed are the hostID and alternatively the *licensePoolId, productID* or *Windows Software ID.* Calling this service releases the license option and returns it to the pool of available license options.

For the complete documentation of opsi service calls see chapter 8.8.

8.5.2. wInst script calls for requesting and releasing of licenses

The wInst provides the client related calls as wInst commands

A wInst script can make a call to the function **`DemandLicenseKey`** to get a license key for installing. The parameters to be passed are:

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

The return value is the license key (a string, can be empty):

```
set $mykey$ = DemandLicenseKey ("pool_office2007")
```

The returned license key can be used by other script command for installing the software.

For releasing a license option and license key (as to be used in a wInst deinstallation script) the command **`FreeLicense`** is available with the following syntax:

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

The boolean function

opsiLicenseManagementEnabled

can be used to decide whether to work with the opsi license management or not:

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

There are some string returning functions available for handling error states, for instance in case no vacant license option is available:

getLastServiceErrorClass

or

getLastServiceErrorMessage

The error handling can be done like this:

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

8.5.3. Manual administration of licensing

Within the opsi config editor, the licenses registered by the opsi service are listed on the tab "Licenses usage":

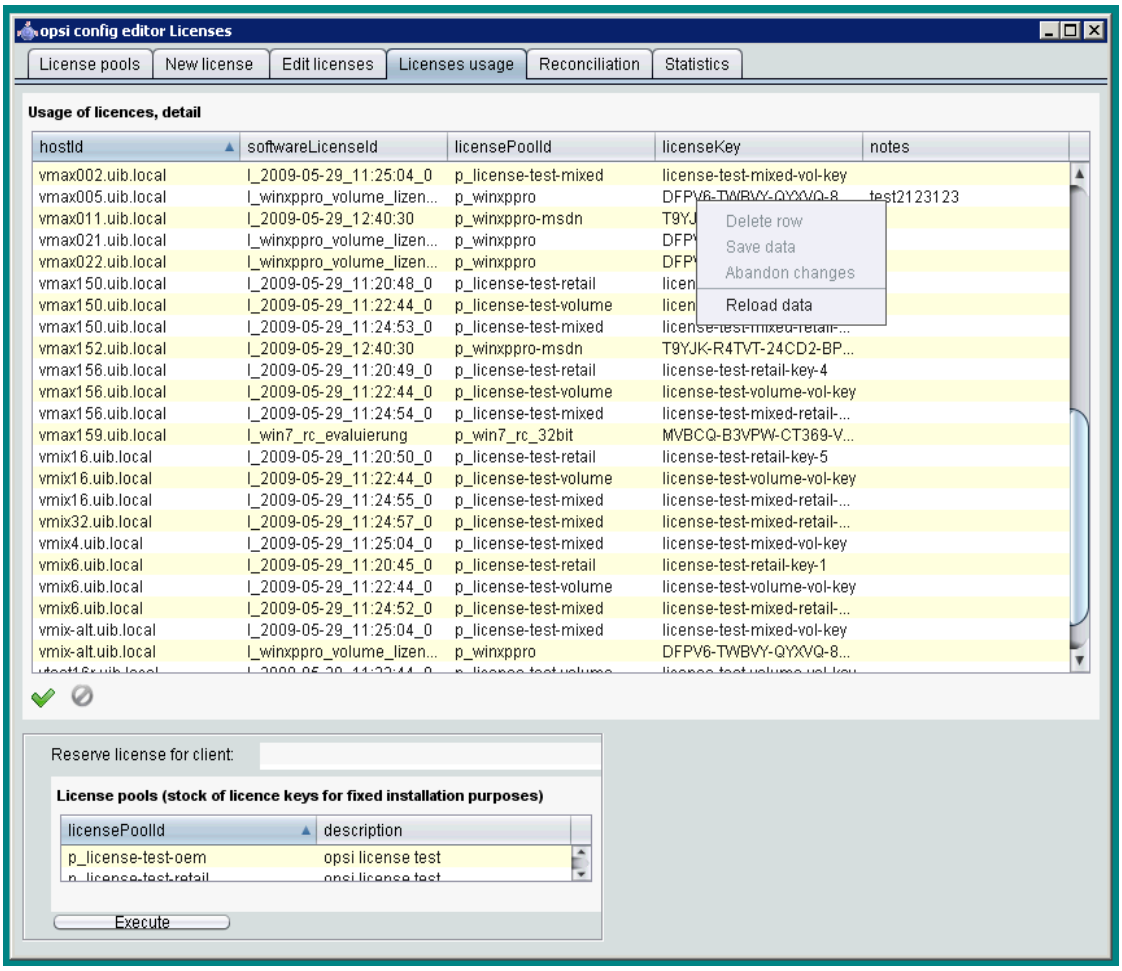


Figure 12: Tab "Licenses usage from the license management window

From this tab, licenses also can be managed manually. This can be useful, if a licensed software is not integrated into the opsi deployment, but installed manually on just a few clients.

These are the functions for manual license management in detail:

- "Delete row" (available from the context menu) releases a license option.
- "Reserve license for client" at the bottom of the window is for making a license reservation for a dedicated client.
- By editing the field "licenseKey" from the "Usage of licenses" table, the license key can be changed.

8.5.4. Preservation and deletion of license usages

If a software packet is reinstalled, the call to winst function `DemandLicenseKey` will return the same license option and license key as had been used before.

In case this is not favored, the former license option has to be released by calling the wInst command `FreeLicense`, or by calling the opsi service call `deleteSoftwareLicenseUsage` or deleting the license use manually.

So, if not explicitly deleted, the license usages are kept when reinstalling a client.

For releasing the licenses, they can be deleted from the tab "Licenses usage" or can be deleted with by a service call

`deleteAllSoftwareLicenseUsages`

passing the client host name to delete the license uses from.

8.6. Reconciliation with the software inventory

The tab "Reconciliation" lists for each client and for each license pool

- whether a use of this license pool is registered by opsi ("used_by_opsi") and
- if the software inventory (swaudit) on that client reported a software, that requires a license option from that pool ("Swinventory_used").

To evaluate the results from swaudit, the relevant Software IDs (as found in the client registry) have to be associated with the appropriate license pool (tab "license pools", see chapter 8.2.2).

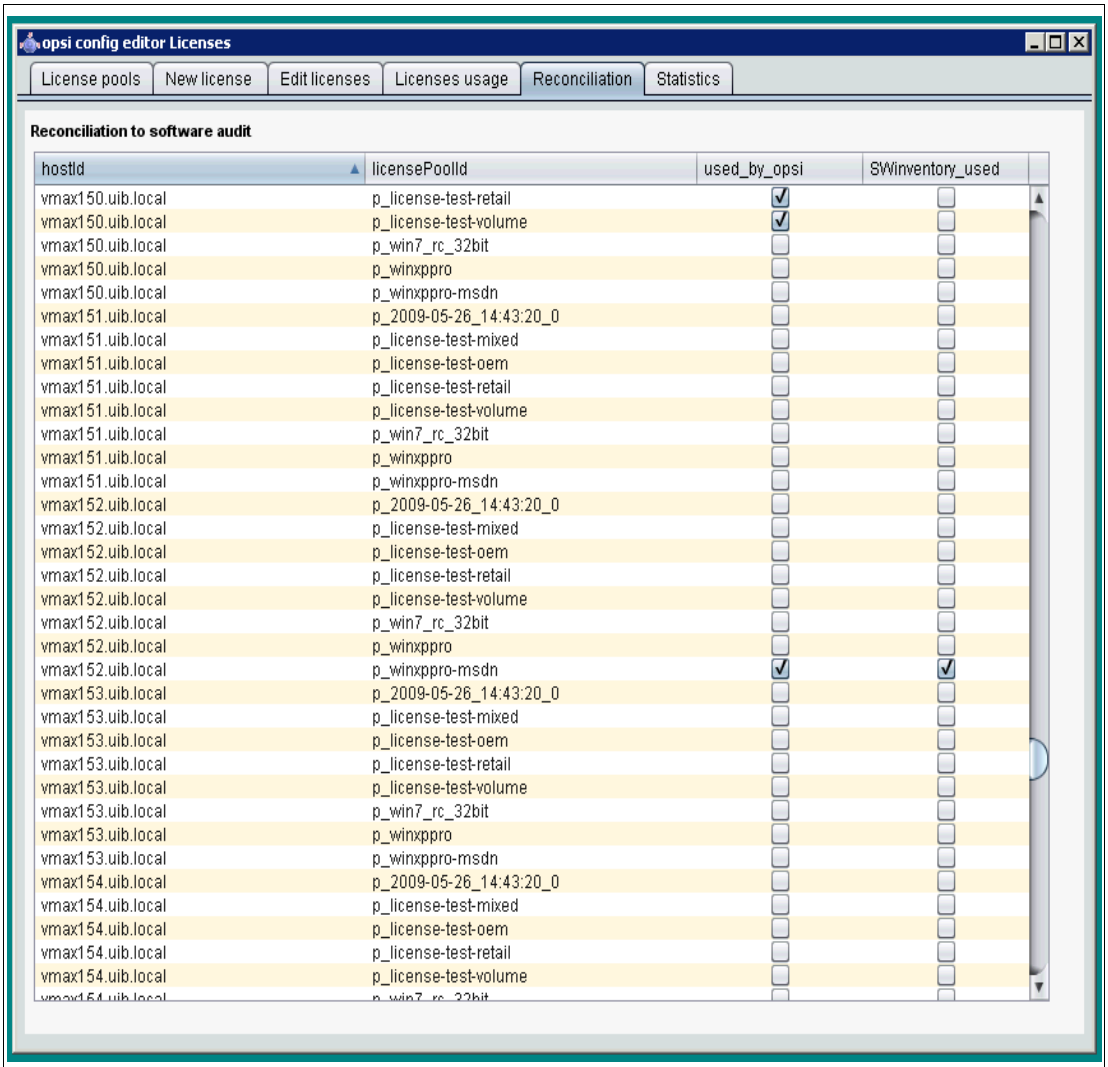


Figure 13: Tab "Reconciliation" from the license management window

8.7. Overlook the license status

The tab "Statistics" is for giving a summary about the different license pools, showing the total number of license options ('license_options') and how many of them are in use ('used_by_opsi') or still available ('remaining opsi').

The number of installations registered by the opsi license management ("used_by_opsi") is listed as well as the number of installations detected by the software audit (resulting in the software inventory). The data from the column 'SWinventory_used' are based on the registry scans from the opsi product 'swaudit' and the assignment of the Windows software IDs (as they are found in the registry) to the license pools (as registered with the opsi license management, see tab "License pools", chapter 2.2).

The context menu provides an option for printing the table.

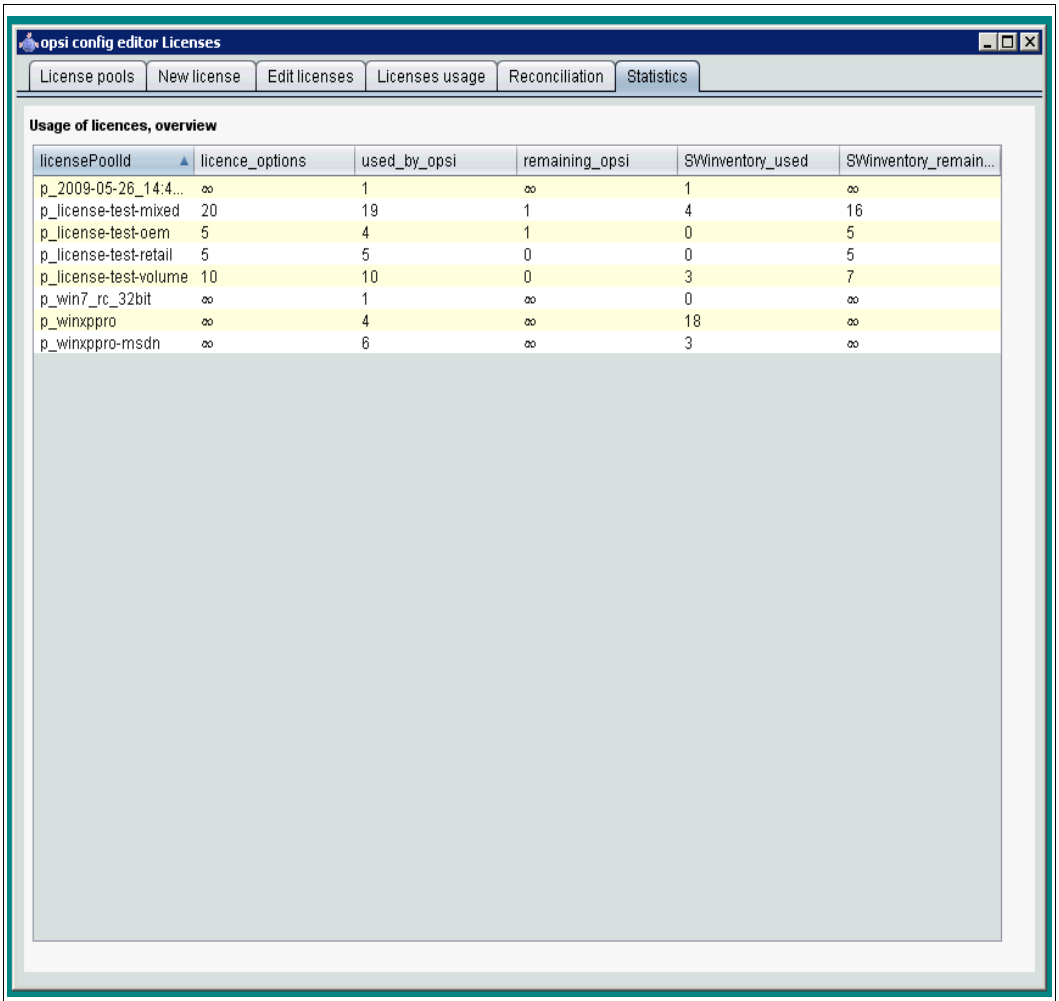


Figure 14: Tab "Statistics" from the licence management window

8.7.1. In case of downgrade option

If a downgrade option has been configured, this appears in the overview and statistics like this:

A single downgrade license results in a license option for at least two different license pools, but only one of them can be requested for an installation. So using a downgrade license option decreases the number of available license options ('remaining_opsi') in each of the license pools concerned by that downgrade option.

So this looks like a single installation reduces the number of available license options twice, which, in this case, actually is the fact.

8.8. Service methods for license management

These service methods can be called for instance from the command line tool 'opsi-admin'. Parameters marked with an asterisk "*" are optional.

8.8.1. Licence contracts

method:
createLicenseContract *(*licenseContractId, *partner, *conclusionDate, *notificationDate, *expirationDate, *notes)*

This method registers a new licence contract record with the ID *licenseContractId*. If no *licenseContractId* is passed, it will be generated automatically. If an existing *licenseContractId* is passed, the existing record will be overwritten.

The parameters *partner* (co-contractor) and *notes* are strings and can be filled with any information desired. The parameters *conclusionDate* (date of conclusion of the contract), *notificationDate* (date for a reminder) and *expirationDate* (expiration date of the contract) are passed in the format YYYY-MM-DD (e.g. 2009-05-18).

Return value is the *licenseContractId* of the new or changed license contract record.

method:
getLicenseContractIds_list ()

This method returns a list of *licenseContractIds* of all existing contract records.

method:
getLicenseContract_hash (*licenseContractId*)

returns a hash with all the information about the license contract identified by ID *licenseContractId*.

The hash keys are:
licenseContractId, *partner*, *notes*, *conclusionDate*, *notificationDate*, *expirationDate*, *softwareLicenseIds*.

Data types of the key values are the same as they are while registering a new record. *softwareLicenseIds* is the list of all software licenses connected with this contract.

method:
getLicenseContracts_listOfHashes ()

returns all the information about all existing contract records as a list of hashes. Each single element of the list is a hash as returned by *getLicenseContract_hash ()*.

method:
deleteLicenseContract (*licenseContractId*)

deletes the contract identified by ID *licenseContractId*. A contract record can only be deleted, if no software licenses are connected to it.

8.8.2. Licenses (software licenses)

method:

createSoftwareLicense (**softwareLicenseId, *licenseContractId, *licenseType, *maxInstallations, *boundToHost, *expirationDate*)

This method creates a new license. If no *softwareLicenseId* is passed, it will be created. When passing an existing *softwareLicenseId ,* the existing record will be changed.

The *licenseContractId* passed assigns the license to a contract. If no *licenseContractId* is passed, the license is assigned to the *default* contract. If this doesn't exist, it will be created.

Available types of license (*licenseType*) are "OEM", "VOLUME" und "RETAIL". If no *licenseType* is passed, it defaults to VOLUME. The maximum number of installations available based on this license is set as *maxInstallations*. If no value or a value < 1is passed for *maxInstallations,* the maximum number of installations is unlimited.
For *licenseType* “OEM“ *maxInstallations* is set to 1.

A license can be assigned to a special host with *boundToHost*. For registering a new record of *licenseType* “OEM“ the parameter *boundToHost* is required.

The optional parameter *expirationDate* sets the limit of validity of the license and is passed as a date in the format YYYY-MM-DD. On the date determined the license runs out f validity.

The method returns the *softwareLicenseId* of the new or changed record.

method:
getSoftwareLicenseIds_list ()

This method returns a list of all registered *softwareLicenseIds,* which is a list of all existing licenses.

method:
getSoftwareLicense_hash (*softwareLicenseId*)

returns all the information about the license with license ID *softwareLicenseId* as a hash.

The hash keys are: *softwareLicenseId*, *licenseContractId*, *licenseType, maxInstallations, boundToHost, expirationDate, licensePoolIds, licenseKeys*.

Data types of the key values are the same as they are while registering a new record. *licensePoolIds* is the list of all software license pools connected with this license.
licenseKeys again is a hash with each key of the hash representing a license pool (*licensePoolId*) the license is connected to. The corresponding value for the key *licensePoolId i*s the *license key* from that license pool.

method:
getSoftwareLicenses_listOfHashes ()

returns the information about all existing software licenses as a list of hashes. Each element of the list is a hash as returned by *getSoftwareLicense_hash ()*.

method:
deleteSoftwareLicense (*softwareLicenseId, *removeFromPools*)

deletes the license with ID *softwareLicenseId*. A license can be deleted only if not used by a client and not assigned to a license pool. The optional parameter

removeFromPools (boolean, default=false) removes the license from all license pools before deletion.

8.8.3. License pools

method:

createLicensePool (**licensePoolId*, **description*, ***productIds*, ***windowsSoftwareIds*)

This method creates a record for a new license pool. If no *licensePoolId* is passed, it will be created. Passing the *licensePoolId* of an existing license pool will change the existing record.

The String *description* can set any description for that license pool.

The parameter *productIds* defines opsi products connected to that license pool, whereas *windowsSoftwareIds* defines a list of Windows Software IDs connected to the license pool.

The method returns the *licensePoolId* of the new or changed license pool record.

method:
getLicensePoolIds_list ()

This method returns the list *licensePoolIds* of all existing licence pool IDs.

method:
getLicensePool_hash(*licensePoolId)*
returns all the information about the license pool with ID *licensePoolId* as a hash.

The hash keys are: *licensePoolId*, *description*, productIds*, windowsSoftwareIds*.

Data types of the key values are the same as they are while registering a new licence pool record.

method:
getLicensePools_listOfHashes ()

returns the information about all of the existing licence pools as a list of hashes. Each element of the list is a hash as returned by *getLicensePool_hash.*

method:´
deleteLicensePool (*licensePoolId, *deleteLicenses*)

deletes the license pool identified by ID *licensePoolId*. A license pool can be deleted only if no software licenses are connected to it. Passing the optional parameter *deleteLicenses* (boolean, default=false) deletes all of the licenses connected to that pool and also the license usages (which is the license connection to the clients).

method:
addSoftwareLicenseToLicensePool (*softwareLicenseId*, *licensePoolId*, **licenseKey*)

connects the software license identified by *softwareLicenseId* to the license pool identified by *licensePoolId.* The optional parameter *licenseKey* sets the license key for that licence option.

method:
removeSoftwareLicenseFromLicensePool (*softwareLicenseId*, *licensePoolId)*

deletes the software license identified by *softwareLicenseId* from the license pool identified by *licensePoolId*.

method:
addProductIdsToLicensePool (*productIds*, *licensePoolId*)

connects a list of opsi product IDs (*productIds*) to the license pool identified by *licensePoolId*.

method:
removeProductIdsFromLicensePool (*productIds*, *licensePoolId*)

Deletes the assignment of opsi products (*productIds*) to the license pool identified by *licensePoolId*.

method:
setWindowsSoftwareIdsToLicensePool (*windowsSoftwareIds*, *licensePoolId*)

assigns a list of Windows software IDs (*windowsSoftwareIds)* to the license pool identified by *licensePoolId*. Existing assignments will be deleted.

method:
getLicensePoolId (**productId*, **windowsSoftwareId*)

returns the license pool ID (*licensePoolId)* connected with the opsi product identified by *productId* or with the Windows software ID *windowsSoftwareId*.

method:
getSoftwareLicenses_listOfHashes (**licensePoolId*)

returns a list of all of the existing software licenses. By passing the optional parameter *licensePoolId* the list returns only licenses connected to this license pool. The elements of the list returnde are hashes with the keys: *softwareLicenseId*, *licensePoolId*, *licenseKey.*

method:
getOrCreateSoftwareLicenseUsage_hash (*hostId*, **licensePoolId*, **productId*, **windowsSoftwareId*)

Returns information about the license use of a dedicated client and license pool. The license pool can be identified by *licensePoolId* or indirectly by passing a *productId* or *windowsSoftwareId.* When specifying an indirect reference by *productId* or *windowsSoftwareId,* the connection to a single license pool must be non-ambiguous. If the connection is ambiguous, the method will throw an exception of type *LicenseConfigurationError.*

If the client has no license from license pool is use, an available license from that pool is assigned to the client. If no license is available, the method throws an exception of type *LicenseMissingError.*

The method returns a hash with the keys *softwareLicenseId*, *licenseKey*, *licensePoolId*, *hostId* and *notes*.

method:
getAndAssignSoftwareLicenseKey (*hostId*, **licensePoolId*, **productId*, **windowsSoftwareId*)

This method is quite the same as *getOrCreateSoftwareLicenseUsage_hash* with the difference, that the license key (*licenseKey*) only is returned.

Another difference is the behavior in case of no available license. If a *productId* had been passed when calling the method, the method checks whether there is a product property named "productkey", which is not an empty string. In this case no exception is thrown but this "productkey" is returned.

method:
getSoftwareLicenseUsages_listOfHashes (**hostIds*, **licensePoolIds*)

This method returns a list of hashes holding information about license usages. The list can be limited by passing the optional parameters *hostIds* (a list of clients) and *licensePoolIds* (a list of license pools).

The method returns a list of hashes with the keys *softwareLicenseId*, *licenseKey*, *licensePoolId*, *hostId* and *notes.*

method:
setSoftwareLicenseUsage (*hostId*, *licensePoolId*, *softwareLicenseId*, **licenseKey*, **notes*)

This method marks a license as being used by a client. If this assignment already exists, it will be modified. The license pool must be passed as *licensePoolId* and the software license as *softwareLicenseId.* The optional parameter *licenseKey* sets the license key being used by the client. The optional parameter *notes* adds some notes.

method:
deleteSoftwareLicenseUsage (*hostId*, **softwareLicenseId*, **licensePoolId*, **productId*, **windowsSoftwareId*)

This method releases the software licenses used by the client *hostId*. The licenses to be released can be limited to a given license pool identified by *licensePoolId or i*ndirectly by passing a productId or windowsSoftwareId (as with *getOrCreateSoftwareLicenseUsage_hash*). By passing a *softwareLicenseId* this license only will be released, otherwise all matching licenses will be released.

method:
deleteAllSoftwareLicenseUsages (*hostIds*):

All software licenses in use by the clients identified by *hostIds* will be released.

method:
getLicenseStatistics_hash (*licensePoolId*)

This method returns information about the use of software licenses from the license pool identified by *licensePoolId*. The hash returned has got the keys:

licenses: number of licenses in the license pool

usedBy: list of clients using a license from the pool.

usageCount: number of clients using a license from the pool.

maxInstallations: maximum total number of installations allowed by that license.

remainingInstallations: Number of available licenses (licenses from that pool that are currently not in use).

8.8.4. Examples for using the methods from scripts

All the methods listed can be called from the command line tool opsi-admin (see the respective chapter of the opsi manual) within a script to, for instance, read existing licenses from a file.

Some examples can be found in the product "license-test-*.opsi" at
http://download.uib.de/opsi3.4/produkte/license-management-test/.
When installing this packet with the opsi-package-manager
  opsi-package-manager -i *.opsi
at "/opt/pcbin/install/<product name>" the script
  create_license-*.sh
is to be found.

Here as an example the script "create_license-mixed.sh".
If in any doubt, better use the download examples mentioned above, for they are probably more recent than the examples listed in this manual:

```
#!/bin/bash
# This is a test and example script
# (c) uib gmbh licensed under GPL

PRODUCT_ID=license-test-mixed
# read the license key from a file
# myretailkeys.txt has one licensekey per line
MYRETAILKEYS=`cat myretailkeys.txt`
# myoemkeys.txt has one pair: <licensekey> <hostid.domain.tld> per line
MYOEMKEYS=`cat myoemkeys.txt`
```

```
# some output
echo "$PRODUCT_ID"

# this is the function to create the oem licenses
##############
createlic ()
{
while [ -n "$1" ]
do
      #echo $1
      AKTKEY=$1
      shift
      #echo $1
      AKTHOST=$1
      shift
      echo "createSoftwareLicense with oem key: ${PRODUCT_ID}-oem-${AKTKEY}
for host ${AKTHOST}"
      MYLIC=`opsi-admin -dS method createSoftwareLicense "" "c_$PRODUCT_ID"
"OEM" "1" "${AKTHOST}" ""`
      opsi-admin -d method addSoftwareLicenseToLicensePool "$MYLIC"
"p_$PRODUCT_ID" "${PRODUCT_ID}-oem-${AKTKEY}"
done
}
##############

# here the script starts

# delete the existing license pool and all connected licenses
# ATTENTION: never (!) do this on a productive system
echo "deleteLicensePool p_$PRODUCT_ID"
opsi-admin -d method deleteLicensePool "p_$PRODUCT_ID" true

# delete the existing license contract
echo "deleteLicenseContract c_$PRODUCT_ID"
opsi-admin -d method deleteLicenseContract "c_$PRODUCT_ID"

# create the new license pool
# the used method has the following syntax:
# createLicensePool(*licensePoolId, *description, *productIds,
*windowsSoftwareIds)
echo "createLicensePool p_$PRODUCT_ID"
opsi-admin -d method createLicensePool "p_$PRODUCT_ID" "opsi license
test" \'['"'$PRODUCT_ID'"']\' \'['"'$PRODUCT_ID'"']\'

# create the new license contract
# the used method has the following syntax:
# createLicenseContract(*licenseContractId, *partner, *conclusionDate,
*notificationDate, *expirationDate, *notes)
echo "createLicenseContract c_$PRODUCT_ID"
opsi-admin -d method createLicenseContract "c_$PRODUCT_ID" "uib gmbh" "" "" ""
"test contract"

# create the new license and add the key(s)
# the used methods have the following syntax:
# createSoftwareLicense(*softwareLicenseId, *licenseContractId, *licenseType,
*maxInstallations, *boundToHost, *expirationDate)
# addSoftwareLicenseToLicensePool(softwareLicenseId, licensePoolId,
*licenseKey)

# create the retail licenses:
```

```
for AKTKEY in $MYRETAILKEYS
do
      echo "createSoftwareLicense with retail key: ${PRODUCT_ID}-retail-$
{AKTKEY}"
      MYLIC=`opsi-admin -dS method createSoftwareLicense "" "c_$PRODUCT_ID"
"RETAIL" "1" "" ""`
      opsi-admin -d method addSoftwareLicenseToLicensePool "$MYLIC"
"p_$PRODUCT_ID" "${PRODUCT_ID}-retail-${AKTKEY}"
done

# create the oem licenses
createlic $MYOEMKEYS

# create the volume licenses
echo "createSoftwareLicense with volume key: ${PRODUCT_ID}-vol-key"
MYLIC=`opsi-admin -dS method createSoftwareLicense "" "c_$PRODUCT_ID" "VOLUME"
"10" "" ""`
opsi-admin -d method addSoftwareLicenseToLicensePool "$MYLIC" "p_$PRODUCT_ID"
"${PRODUCT_ID}-vol-key"#
```

8.9. Example products and templates

In the uib download section at

http://download.uib.de/opsi3.4/produkte/license-management-test/

there are four example products. One for each type of license model, as there are Retail, OEM and Volume license type, as well as a product combining all of them.

These example products use as an example some licenses and release them again. So using them leaves some marks in the software inventory, that might be of influence to reconciliation and statistics.

All of these products contain a shell script to automatically generate license pools, license contracts and license options.

The standard template for Winst scripts also contains some examples for using the opsi license management.

9. opsi-Module: depot server

9.1. Overview

The opsi depot server is a special server installation based on GNU/Debian Linux. It is the base installation for the modules 'software distribution' and 'operating system installation'.

For the software distribution it provides secured file shares, where configuration files and software packets (software depots) are protected against unauthorized access. The password transmission to the client for connecting these shares is encrypted, so just the modules of the software distribution and the system administrator get access to these shares.

The opsi depot server comes with a web interface for opsi configuration and abstraction layer for the data backend.

Another important module of the opsi depot server is to supply of services for the automated operating system installation:

- dhcp for the administration of IP addresses,
- tftp for the transmission of bootimages and configuration information.

In addition some interactive and scrip-table tools for the administration of the configuration files and the boot images are provided.

For security reasons, stability and minimum resource consumption the opsi depot server is limited to the reasonable and so the hardware requirements are very low. The opsi depot server can also run as a virtual instance, e.g. vmware® (www.vmware.com).

9.2. Installation and initial operation

The installation media to install an opsi depot server can be downloaded from the opsi website.

9.3. Access to the graphic user interface of the depot server via VNC

Opsi depot server has no X-server for special hardware. The console of the depot server is plain text therefore. As X-server the (tight) VNC-server is in operation. So using vncviewer the X-server running on the opsi depot server is accessible from any computer in your network. This structure (without any hardware specific adaptions) allows for good standardization, which simplifies maintenance and increases stability.

At starting the depot server a VNC-server is started for every administrator, who is registered in the file /etc/vncuser. If for any reason the VNC-server isn't running, it can be started from the command prompt as '**vnc-server**'.

Configuring the VNC-Servers:

The file /etc/vncusers is the configuration file for the VNC-Server. For every user a „default“ VNC-Server can be set. This VNC-Server will start when the booting the opsi depot server.

```
#parameters for vncserver
#
#examples:
#
#username:displayno:resolution:start_at_boot:localhost:
#test2:45:800x600:start_at_boot:localhost:
#test3:46:800x600::: # mandatory entries
#
#start_at_boot and localhost can be omitted, but not the colons!
#displayno should be different for each entry
root:49:1260x960:start_at_boot::
```

The file has a similar structure as the /etc/passwd (important is the number of colons for they are field delimiters).

1. Field: user name
2. Field: this is the display number, the port the VNC-server listens to (1-99 allowed)
3. Field: display resolution. For a 1024x786 display will 1050x700 be a good choice.
4. Field: start_at_boot, only with this text at the boot of the opsi depot server the VNC-server will be started automatically; (usually not necessary).
5. Field: localhost, the VNC-server can be connected from localhost only. This can be used for ssh tunneling.

Being logged on as a user at the opsi depot server, the vncserver specified for that user can be started by '**vncserver**'.

The first start of vncserver prompts for a password, which will be saved in '~USER/.vnc/passwd' and will be valid for all VNC-servers of this user.

To stop the VNC-server:

```
vncserver -kill :<DisplayNumber>
```

What is VNC ?

VNC is open source software distributed under the GNU Public License.

VNC (Virtual Network Computing) is an (almost) operating system-independent client/server application, which provides the graphical user interface of a specified computer (= server) on the own desktop (= Client) to work with this computer remotely. VNC consists of a server and a client application, which can be configured according to different purposes.
Regarding to the VNC naming convention, the 'client' is the local computer you are sitting at. The 'server' is the remote computer, which is to be accessed via network.

The remote server application exports its display to the local client application and also provides an interface for keyboard and mouse input. For security reasons the server application should be configured for using passwords, which have to be passed at the connection request.

Websites for vnc : http://www.realvnc.com/
and tightvnc: http://www.tightvnc.com/

9.4. Shares for software packets and configuration files

9.4.1. Samba Configuration

The opsi depot server provides network shares holding the configuration information and the software packets. These shares can be mounted by the clients. For Windows Clients the shares are provided by SAMBA (version 3.x).

On the opsi depot server the SAMBA configuration files are located in '/etc/samba'. The file '/etc/samba/smb.conf' holds the general settings and configurations. The specifications of the shares are also located in 'smb.conf' or in the include file

'share.conf'. In this file it is configured, what shares are provided for the modules 'software depot server', 'utilities server' and 'configuration management server'. There can be a different share for each module, but the default setting is, that all of these modules are using a single share. The default setting is to share the directory '/opt/pcbin' as the SAMBA share 'opt_pcbin'. Any changes to these defaults have to be configured in the file '/etc/samba/share.conf' and also in the global opsi network configuration. Changing the SAMBA configuration, a SAMBA reload is required for the changes to come into effect (/etc/init.d/samba reload).

In principle the SAMBA 3.x installation of the opsi depot server can be extended to be a a fully featured file- and print server. Uib gmbh offers comprehensive support in this area.

Example for the share.conf:

```
[opt_pcbin]
 available = yes
 comment = opsi depot share
 path = /opt/pcbin
 oplocks = no
 level2 oplocks = no
 writeable = yes
 invalid users = root

[opsi_config]
 available = yes
 comment = opsi config share
 path = /var/lib/opsi/config
 writeable = yes
 invalid users = root

[opsi_workbench]
 available = yes
 comment = opsi workbench share
 path = /home/opsiproducts
 writeable = yes
 invalid users = root
```

9.4.2. Required administrative user accounts and groups

9.4.2.1. User opsiconfd

The opsiconfd deamon is started as user 'opsiconfd'.

9.4.2.2. User pcpatch

Access to the client configuration files and the software depot should be restricted and should be granted only to the system administrators and the software distribution service (opsi preLoginLoader) running on the client. This for instance is required to meet the license agreements, which is in the system administrators responsibility.

To allow this the user account 'pcpatch' gets the user-ID 992, the home directory '/opt/pcbin/pcpatch' and as default the password 'Umwelt'. Change the password with:
`opsi-admin -d task setPcpatchPassword`
The user 'pcpatch' is the owner of the opsi configuration files and on this account run the opsi service processes. Also the opsi-PreLoginLoader uses this account for connecting the depot server shares.

9.4.2.3. Group pcpatch

Beside the user 'pcpatch' there is also a group 'pcpatch'. The user 'pcpatch' as well as the group 'pcpatch' has full access on most of the opsi files. All the administrators of the opsi depot server should therefor be member of the group 'pcpatch', so they have write access to the configuration data.

A user can join group 'pcpatch' by: '**`addgroup`** <user> **`pcpatch`**'.

9.4.2.4. Group opsiadmin

Members of the group 'opsiadmin' are permitted to connect the opsi-webservice and can use for instance the 'opsi-configed' configuration editor. Therefor all opsi administrators should be members of the group 'pcpatch'.

A user can join group 'opsiadmin' by: '**`addgroup`** <user> **`opsiadmin`**'.

9.4.3. Depot share with software packets (install)

The depot-Share provides all the software-packets which are installable by the client task 'wInst'. The default directory for the software packets is the directory '/opt/pcbin/install'. In this directory each software packet has its own sub directory

named as the software packet. These sub directories contain the packet-specific installation scripts and files.

9.4.4. Config share with configuration and logging (pcpatch)

When using the 'file' backend, the client configuration files are located in the configuration-share, one file per client. This directory is preset to '/opt/pcbin/pcpatch' on the opsi depot server. In the sub directory 'pclog' are the client-specific log files: log file from the OS installation (e.g. hardware information, hard disk partition info) and the error logs from the client software installation. Using the 'File31' backend all the configuration data is located in '/var/lib/opsi/config'.

9.4.5. Utils share: Utilities (utils)

The utils-share contains several opsi client utilities. The directory is preset to '/opt/pcbin/utils' on the opsi depot server.

9.5. Administration of PCs via DHCP

9.5.1. What is DHCP?

DHCP is part of the TCP/IP protocol stack to exchange and set information about the network configuration and components between client and server.

The *DHCP* protocol can be seen as an extension of the older *BOOTP* protocol. It allows the dynamic allocation of IP addresses for client PCs (this DHCP feature is not used with the opsi depot server).

For client PCs most of the common network controllers can be used if they have a bootprom:

- Network controllers with *PXE-bootprom* (= Preboot Execution Environment)
- Network controllers with older *TCP/IP BOOTP-bootprom* (e.g. bootix bootproms).

The IP address of a PC-client can be found in the '/etc/hosts'.

The other configuration data is located in the file '**/etc/dhcp3/dhcpd.conf**'. This file can be edited (in addition to the common unix/linux editors) web based with the gui-tool **webmin**(web based interface for system administration for Unix).

Basically there are three ways of IP address allocation on DHCP-Servers:

> **Dynamically:** From within a certain range of IP addresses vacant addresses are assigned to a client for a certain amount of time. At expiration – even during a working-session – the client has to try to extend this assignment, but eventually the client gets a new IP address. In this way the same IP address can be used at different times by different clients.
>
> **Automatically:** An unused IP address is assigned to each client automatically for an unlimited time.
>
> **Manually:** The assignment of the IP addresses is configured by the system administrators manually. At a DHCP-request this address is assigned to the client. For the opsi depot server this method is recommended, since this simplifies the network administration.

PCs with a static IP address can use both protocols DHCP/PXE or BOOTP (depends on the network controller's bootprom).

A dynamic or automatic IP address assignment can only be realized with DHCP and PXE bootproms.

BOOTP (Bootstrap Protocol) only supports static assignment of MAC and IP addresses, like the manual assignment with DHCP.
There are only 2 types of data packets with BOOTP: **BOOTREQUEST** (Client-Broadcast to Server = request for IP address and boot parameters to a server) and **BOOTREPLY** (Server to Client: advise of IP address and boot parameters).

At the start of the network connection the only information a network controller has got is its own hardware address (= hardware Ethernet, MAC of the NIC), consisting of six two-digit hexadecimal numbers.

The PXE firmware is activated at boot time and sends a **DHCPDISCOVER** broadcast request into the network (standard port). It is a request for an IP address and for the name of the DHCP server in charge.

With **DHCPOFFER** the DHCP-Server makes a proposal.

DHCPREQUEST is the client's answer to the server, if the offered IP address is accepted (there might be several DHCP servers in the network).

With **DHCPACK** the DHCP server acknowledges the client request and sends the requested information to the client.

Additional data packets:

- DHCPNACK — Rejection of a DHCPREQUEST by the Server.
- DHCPDECLINE — Rejection by the Client, because the offered IP address is already in use
- DHCPRELEASE — The client releases the IP address (so it is available for a new assignment).
- DHCPINFORM — Client request for parameters (but not for an IP address).

9.5.2. Dhcpd.conf

The opsi depot servers 'dhcpd.conf' is limited to just the required information and functions:

- PC name,
- hardware Ethernet address,
- IP address of the gateway,
- net mask,
- IP address of the boot server,
- name of the boot file,
- URL of the OPSI configuration files.

Internal structure of the 'dhcpd.conf'

Lines with configuration instructions are terminated by a semicolon (;). Empty lines are allowed. Comments begin with a hash(#) (the „host description“, an additional description for the PC in front of the host name, is realized in the same way.).

At the beginning of the '/etc/dhcp3/dhcpd.conf' are some general parameters. In the second part the entries for subnets, groups and hosts are located. A hierarchical grouping of clients can be done by enclosing entries (e. g. subnet and group) with curly brackets. The defaults of a block refer to all elements within this block.

General parameters / example

```
# Sample configuration file for ISC dhcpd for Debian
# also answer bootp questions
allow bootp;
```

The network protocol bootp is supported.

PC-specific entries

A DHCP-configuration file must have at least one subnet definition. Everything defined within the brackets is valid for all hosts or groups of that subnet.
By the element 'group' groups of computers can be defined, which have common parameters (so the common parameters do not have to be defined for every client).
If different instructions are set on different levels, then the innermost definition overwrites the outer one.

```
subnet{
.....
        group{
        .....
                host{
                .....
                }
        }
}
```

Example

```
# Server Hostname
```

```
server-name "schleppi";
subnet 194.31.185.0 netmask 255.255.255.0{
      option routers 194.31.185.5;
      option domain-name "uib.net";
      option domain-name-servers 194.31.185.14;
#Group the PXE bootable hosts together
   group {
```

Here is the beginning of a group of PCs within a subnet;
Example: Group of PCs with PXE-Network-Interface-Controllers.

```
  # PXE-specific configuration directives...
    # option dhcp-class-identifier "PXEClient";
    # unless you're using dynamic addresses
    filename "linux/pxelinux.0";
```

All PCs within this group use a Linux bootfile, unless something different is defined in the PC-entry.

```
                host pcbon13 {
                hardware ethernet 00:00:CB:62:EB:2F;
                        }
```

This entry only contains hostname and hardware address (MAC).
The hardware address is six couples of hexadecimal characters (not case sensitive), which must be separated by a colon!

```
  }
}
```

The curly brackets mark the end of the segments 'group' and 'subnet'.

If a new PC should join the subnet, it has to be registered in 'dhcpd.conf'.

After changing the DHCP configuration file, the DHCP server must be restarted, so that the new configuration is applied to the DHCP server:
/etc/init.d/dhcp3-server restart

9.5.3. Tools: DHCP administration with Webmin

Since the syntax of the 'dhcpd.conf' is quite complex, the depot server provides a graphical web based tool for DHCP administration. The well known administration tool 'webmin' is used as a graphical interface to the 'dhcpd.conf'.
The service 'webmin' should be running after booting the server. Otherwise it can be started (as user root) by: '**`/etc/init.d/webmin start`**'.

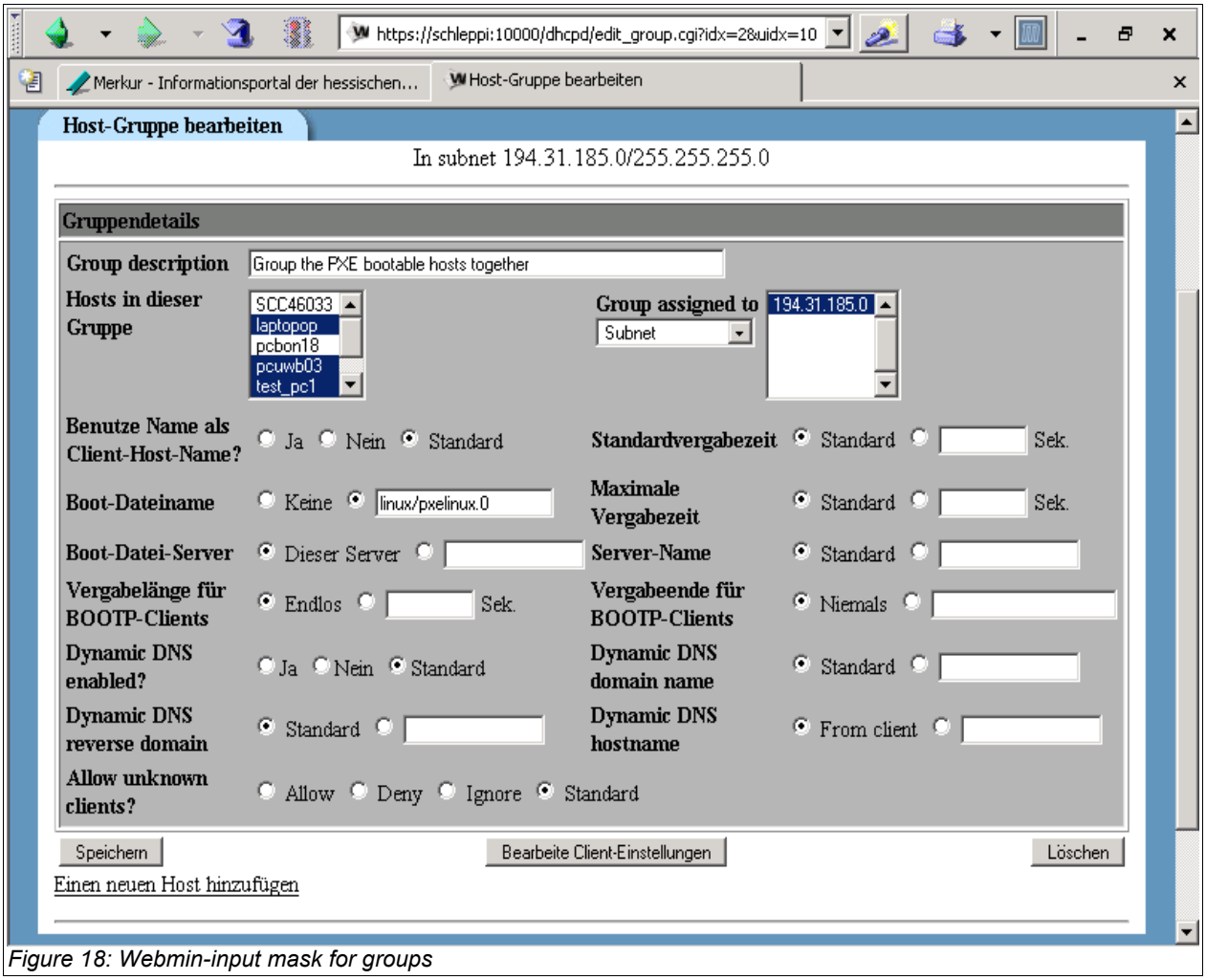


Figure 18: Webmin-input mask for groups

If 'webmin' is running on the server, it can be connected from any client by: 'https://<server name>:10000' (default user/password: admin/linux123).

9.6. opsi V3: opsi configuration API, opsiconfd and backend manager

Opsi V3 comes with a python based configuration API. This API provides an abstraction layer interface to the opsi configuration, which is independent of the actual type of backend in use. Also there are some internal functions to operate on a special type of backend. The backend manager configuration file (/etc/opsi/backendmanager.conf) defines which backend type is to use.

The tool 'opsi-admin' provides a command line access to the configuration-API. In the corresponding chapter you get a detailed overview of the API functions.

In addition the opsi server provides a web service as an interface to the API to be connected by other tools and services (for example the graphical configuration tools, opsi-wInst or opsi bootimage). The web service isn't based on XML/Soap but on the compact JSON standard (www.json.org). The web service is part of the opsi configuration daemon 'opsiconfd'. The web service can be connected via https through port 4447 and also provides a simple interactive web GUI.

The 'opsiconfd' runs as user 'pcpatch'. So the user 'pcpatch' since opsi V3 needs different privileges as with opsi V2.

The configuration file for 'opsiconfd' is '/etc/opsi/opsiconfd.conf'.

The 'opsiconfd' log files are written to '/var/log/opsi/opsiconfd', a separate file for each client.

10. opsi-server with multiple depots

10.1. Support

The functions described in this chapter are complex and you will get only support on this topic via a professional support contract.

10.2. Concept

Supporting multiple depotshares in opsi aims at the following targets:

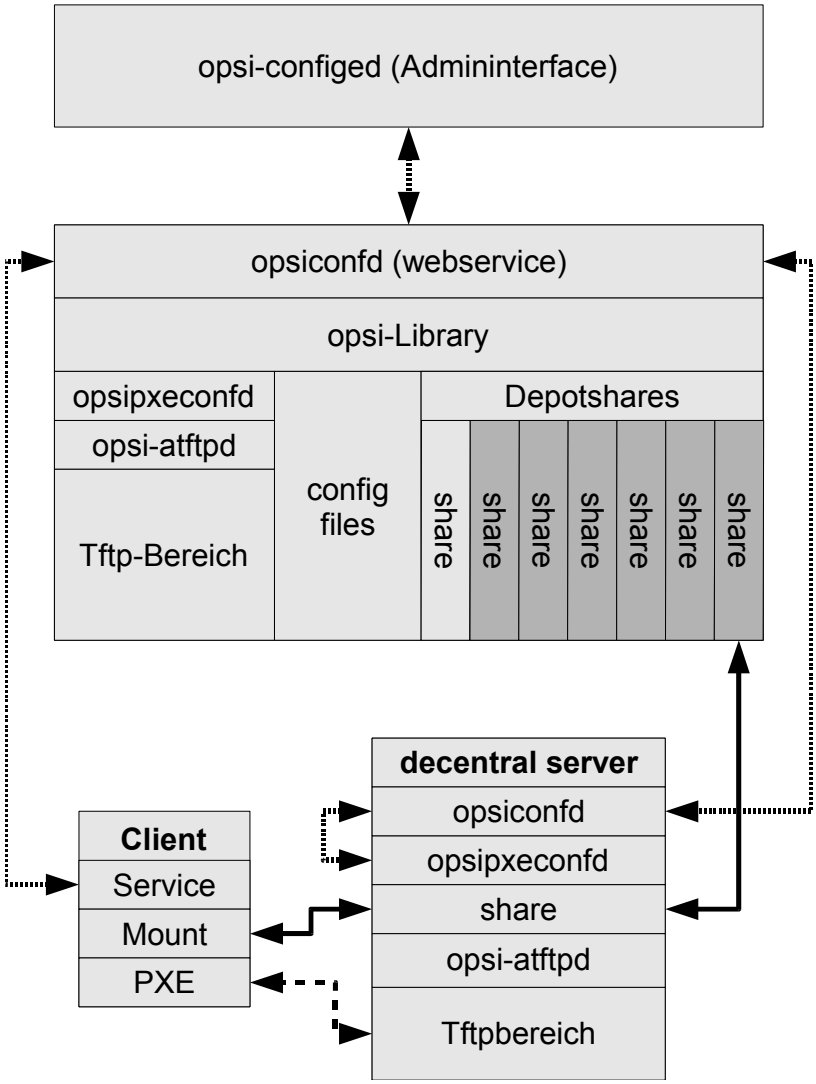
- central configuration data storage and configuration management
- providing the software depots on local servers
- automated deployment of software packages from the central server to the local depots

Accordingly, it is implemented:

- All configuration data is stored on the central opsi-config-server
- All clients connect to this config-server in order to request their configuration data. The configuration data comprise the information on method and target of the depot-server connection.
- All installable software is stored on depot-servers.
- The depot-servers have as well an opsipxeconfd running by which they provide bootimages to clients via PXE/tftp.

10. opsi-server with multiple depots

Schema: opsi with a decentral depot-server



- opsi-package-manager. A program to (de-)install opsi packages on one ore more depot-servers.
- The opsi packages are copied via webdav protocol to the depot-servers and are installed from the opsiconfd via a webservice call.
- opsi-configed supports the management of multiple depots.

- Clients connected to different depots can be managed in one bundle if the involved depots are synchronized (have all product packages in identical versions).

the following schema gives a more detailed view on the communication between the components of a opsi multi depot share environment.

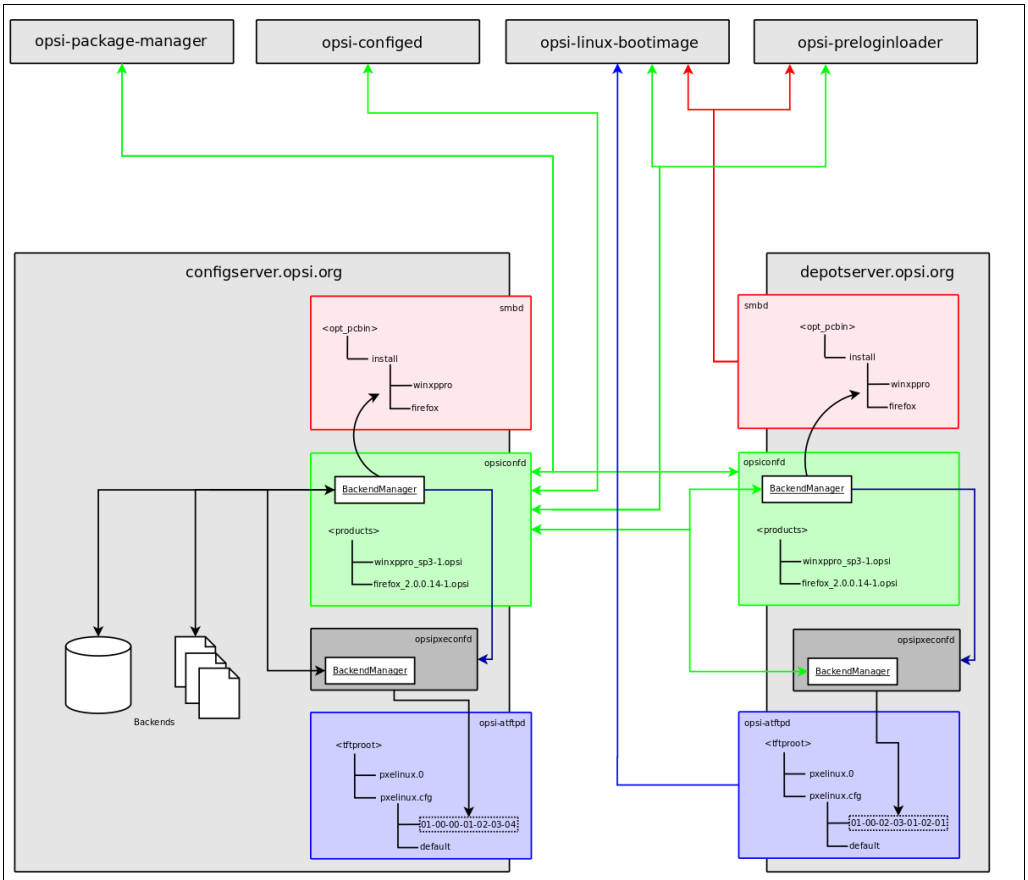


Figure 20: Components and communication of a multi depot installation

10.3. Creating a (slave) depot-servers

In order to create a opsi-depot-server you have to install a standard opsi-server (see opsi-server installation manual). This server can be configured to act as depot-server by calling the script **`/usr/share/opsi/register-depot.py`** . Because this script does not only reconfigure the local server, but also registers this server as depot-server with the central config-server, username and password of a member of the opsiadmin group have to be supplied here.

Example:
vmix12.uib.local will be reconfigured as depot-server and registered with the config-server bonifax.uib.local:

```
vmix12:~# /usr/share/opsi/register-depot.py

*****************************************************************************
***
*         This tool will register the current server as depotserver.
*
* The config file /etc/opsi/backendManager.d/15_jsonrpc.conf will be
recreated. *
*                  =>>> Press <CTRL> + <C> to abort <<<=
*
*****************************************************************************
***

 Config server [localhost]: bonifax.uib.local
 Account name to use for login [root]: oertel
 Account password [password]:
Connecting to host 'bonifax.uib.local' as user 'oertel'
 The subnet this depotserver is resonsible for [192.168.4.0/24]:
 Description for this depotserver [Depotserver vmix12]:
 Additional notes for this depotserver [Notes for vmix12]:

Creating depot 'vmix12.uib.local'
Requesting base-url '/rpc', query '{"params":
["vmix12","uib.local","file:///opt/pcbin/install","smb://vmix12/opst_pcbin/ins
tall","file:///var/lib/opsi/products","webdavs://vmix12.uib.local:4447/product
s","192.168.4.0/24","Depotserver vmix12","Notes for
vmix12"],"id":1,"method":"createDepot"}' failed:
Trying to reconnect...
Testing connection and setting pcpatch password
Connection / credentials ok, pcpatch password set!
Creating jsonrpc backend config file
/etc/opsi/backendManager.d/15_jsonrpc.conf
Patching config file /etc/opsi/backendManager.d/30_vars.conf
```

10.4. packetmangment with the opsi-package-manager

see also chapter 3.4 Tool: opsi-package-manager: (de-)installs opsi-packages on page 28

In or to manage opsi-packages with different depot-servers the opsi-package-manager got the option -d (or --depot). With this option you can give the target depot-server for the Installation. Using the keyword 'ALL' the opsi package will be copied to

/var/lib/opsi/products on all known depot-servers and then installed via a local webservice call.

If you don't give the option -d, the opsi package will be only installed on the local server (without upload to /var/lib/opsi/products)

Example:
Install the package softprod_1.0-5.opsi on all known depot-servers:

```
opsi-package-manager -d ALL -i softprod_1.0-5.opsi
Processing upload of 'softprod_1.0-5.opsi' to depot 'bonifax.uib.local'
Processing upload of 'softprod_1.0-5.opsi' to depot 'vmix13.uib.local'
Processing upload of 'softprod_1.0-5.opsi' to depot 'vmix12.uib.local'
Overwriting destination 'softprod_1.0-5.opsi' on depot 'bonifax.uib.local'
Starting upload of 'softprod_1.0-5.opsi' to depot 'bonifax.uib.local'
   100.00%        3 KB       0 KB/s  00:00 ETAs - softprod_1.0-5.opsi >>
bonifax.uib.local
Upload of 'softprod_1.0-5.opsi' to depot 'bonifax.uib.local' done
Installing package 'softprod_1.0-5.opsi' on depot 'bonifax.uib.local'
Overwriting destination 'softprod_1.0-5.opsi' on depot 'vmix13.uib.local'
Starting upload of 'softprod_1.0-5.opsi' to depot 'vmix13.uib.local'
Overwriting destination 'softprod_1.0-5.opsi' on depot 'vmix12.uib.local'
Starting upload of 'softprod_1.0-5.opsi' to depot 'vmix12.uib.local'
   100.00%        3 KB       3 KB/s  00:00 ETAs - softprod_1.0-5.opsi >>
vmix13.uib.local
   100.00%        3 KB       3 KB/s  00:00 ETAs - softprod_1.0-5.opsi >>
vmix12.uib.local
Upload of 'softprod_1.0-5.opsi' to depot 'vmix12.uib.local' done
Installing package 'softprod_1.0-5.opsi' on depot 'vmix12.uib.local'
Upload of 'softprod_1.0-5.opsi' to depot 'vmix13.uib.local' done
Installing package 'softprod_1.0-5.opsi' on depot 'vmix13.uib.local'
Installation of package '/var/lib/opsi/products/softprod_1.0-5.opsi' on depot
'bonifax.uib.local' finished
Installation of package '/var/lib/opsi/products/softprod_1.0-5.opsi' on depot
'vmix13.uib.local' finished
Installation of package '/var/lib/opsi/products/softprod_1.0-5.opsi' on depot
'vmix12.uib.local' finished
```

In this example three depot-servers are known (bonifax.uib.local, vmix12.uib.local, vmix13.uib.local). The opsi-package-manager first starts the uploading of the package to the depots. When the uploads are finished the installation takes place. The local depot is treated in the same way as the external depots.

In order to get information's about what are the differences between depots you may call opsi-package-manager with the option -D (or --differences).

Example:
Show the differences between all known depots regarding the product mshotfix

```
opsi-package-manager -D -d ALL mshotfix
```

```
mshotfix
    vmix12.uib.local :  200804-1
    vmix13.uib.local :  200804-1
    bonifax.uib.local:  200805-2
```

10.5. configuration files

see Chapter 15.3.1.5 Configuration files in /var/lib/opsi/config/depots/<depotid> on page 140

11. DHCP and name resolving (DNS)

have to be written

12. opsi data storage (backend)

12.1. File backend

With the backend type 'file backend' the configuration information is kept in text files (ini file syntax) on the server.

12.1.1. File3.1-Backend (opsi 3.1)

Basic features of the backend 'File3.1' :

- current opsi default backend
- Linux standard base conform
- not backward compatible with opsi 2.x/3.0
- all opsi functions are available with this backend
- works only for clients which are run in 'service'-mode (accessing their configuration files via opsi service).

The actual data files are kept in '/var/lib/opsi'.

Content and configuration of these files are described in the chapter 10 “Important data files of the opsi depot server”.

12.2. LDAP backend

The opsi-LDAP-schema is saved as '/etc/ldap/schema'.

For activation of the LDAP-Backend a functional LDAP-server has to be accessible.

The opsi LDAP-schema has to be included to the LDAP configuration file '/etc/ldap/lapd.conf':

```
include         /etc/ldap/schema/opsi.schema
```

(the LDAP service 'sldap' has to be restarted)

The next step is to patch the backend configuration of opsi.

12.2.1. Integrating the LDAP-backend

To activate the LDAP-backend change the following settings in '/etc/opsi/backendmanager.conf':

Settings for the file-backend:

```
self.backends[BACKEND_FILE] =           { 'load': True }
self.backends[BACKEND_LDAP] =           { 'load': False }
```

Settings for the LDAP-backend:

```
self.backends[BACKEND_FILE] =           { 'load': False }
self.backends[BACKEND_LDAP] =           { 'load': True }
```

12.2.2. Configuring the LDAP-backend

```
    self.backends[BACKEND_LDAP]['config'] = {
        "host":     "localhost",
        "bindDn":   "cn=admin,%s" % baseDn,
        "bindPw":   "password",
    }
```

12.2.3. Assign the LDAP-backend to methods

```
self.defaultBackend = BACKEND_LDAP
self.clientManagingBackend = [ BACKEND_DHCPD, BACKEND_LDAP ]
self.pxebootconfBackend    = BACKEND_OPSIPXECONFD
self.passwordBackend       = BACKEND_FILE31
self.pckeyBackend          = BACKEND_FILE31
self.swinventBackend       = BACKEND_MYSQL
self.hwinventBackend       = BACKEND_MYSQL
self.loggingBackend        = BACKEND_FILE31
```

In this example the LDAP backend is set as the default backend. The PC-keys and the pcpatch-passwords are still administrated as files.

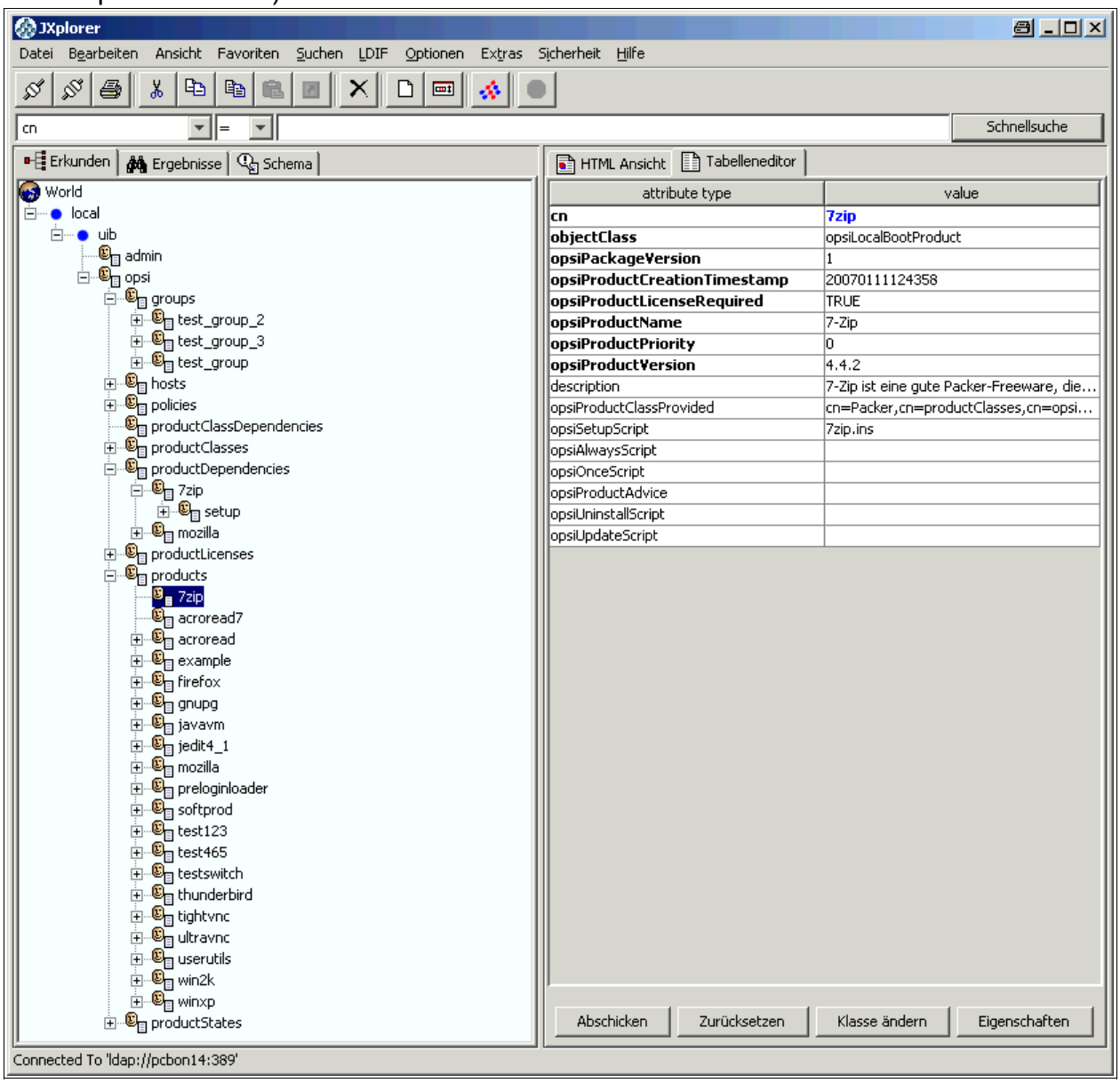
Now restart the opsi config daemon 'opsiconfd':

```
/etc/init.d/opsiconfd restart
```

The following command creates the LDAP base structure:

```
opsi-admin -d method createOpsiBase
```

Underneath the LDAP-base node is an organizationalRole cn=opsi (e.g. cn=opsi, dc=uib, dc=local). You find underneath the node opsi all of the opsi data. This structure can be explored very easily with a graphical frontend like the Jxplorer (which is included in the opsi-adminutils).



12.3. MySQL-backend for inventory data

12.3.1. overview and datastructure

Inventory data is stored in structured text files by default. This type of storage is not very useful if you wish to form free queries on these data. In order to allow free queries and reports a mysql based backend for the inventory data has been introduced.

The main characteristics of this backend are:

- only for inventory data (up to now)
- optional (not the default backend)
- a very fine granulated data structure with an additional table to make queries easier.
- a history function which tracks changes in the inventory.

The MySQL based backend for the inventory data exists since opsi 3.3. Regarding the very different structure of the components in the inventory the resulting datastructure is complex.

The table 'hosts' comprises all known hosts. For every device type we use two tables: The HARDWARE_DEVICE_ .table describes the model without individual aspects like the serial number. The HARDWARE_CONFIG table stores these individual and configuration data.

These both tables are connected via the field hardware_id. This is the resulting list of tables:

```
HARDWARE_CONFIG_1394_CONTROLLER
HARDWARE_CONFIG_AUDIO_CONTROLLER
HARDWARE_CONFIG_BASE_BOARD
HARDWARE_CONFIG_BIOS
HARDWARE_CONFIG_CACHE_MEMORY
HARDWARE_CONFIG_COMPUTER_SYSTEM
HARDWARE_CONFIG_DISK_PARTITION
HARDWARE_CONFIG_FLOPPY_CONTROLLER
HARDWARE_CONFIG_FLOPPY_DRIVE
HARDWARE_CONFIG_HARDDISK_DRIVE
HARDWARE_CONFIG_IDE_CONTROLLER
HARDWARE_CONFIG_KEYBOARD
HARDWARE_CONFIG_MEMORY_BANK
HARDWARE_CONFIG_MEMORY_MODULE
```

```
HARDWARE_CONFIG_MONITOR
HARDWARE_CONFIG_NETWORK_CONTROLLER
HARDWARE_CONFIG_OPTICAL_DRIVE
HARDWARE_CONFIG_PCI_DEVICE
HARDWARE_CONFIG_PCMCIA_CONTROLLER
HARDWARE_CONFIG_POINTING_DEVICE
HARDWARE_CONFIG_PORT_CONNECTOR
HARDWARE_CONFIG_PRINTER
HARDWARE_CONFIG_PROCESSOR
HARDWARE_CONFIG_SCSI_CONTROLLER
HARDWARE_CONFIG_SYSTEM_SLOT
HARDWARE_CONFIG_TAPE_DRIVE
HARDWARE_CONFIG_USB_CONTROLLER
HARDWARE_CONFIG_VIDEO_CONTROLLER
HARDWARE_DEVICE_1394_CONTROLLER
HARDWARE_DEVICE_AUDIO_CONTROLLER
HARDWARE_DEVICE_BASE_BOARD
HARDWARE_DEVICE_BIOS
HARDWARE_DEVICE_CACHE_MEMORY
HARDWARE_DEVICE_COMPUTER_SYSTEM
HARDWARE_DEVICE_DISK_PARTITION
HARDWARE_DEVICE_FLOPPY_CONTROLLER
HARDWARE_DEVICE_FLOPPY_DRIVE
HARDWARE_DEVICE_HARDDISK_DRIVE
HARDWARE_DEVICE_IDE_CONTROLLER
HARDWARE_DEVICE_KEYBOARD
HARDWARE_DEVICE_MEMORY_BANK
HARDWARE_DEVICE_MEMORY_MODULE
HARDWARE_DEVICE_MONITOR
HARDWARE_DEVICE_NETWORK_CONTROLLER
HARDWARE_DEVICE_OPTICAL_DRIVE
HARDWARE_DEVICE_PCI_DEVICE
HARDWARE_DEVICE_PCMCIA_CONTROLLER
HARDWARE_DEVICE_POINTING_DEVICE
HARDWARE_DEVICE_PORT_CONNECTOR
HARDWARE_DEVICE_PRINTER
HARDWARE_DEVICE_PROCESSOR
HARDWARE_DEVICE_SCSI_CONTROLLER
HARDWARE_DEVICE_SYSTEM_SLOT
HARDWARE_DEVICE_TAPE_DRIVE
HARDWARE_DEVICE_USB_CONTROLLER
HARDWARE_DEVICE_VIDEO_CONTROLLER
HARDWARE_INFO
HOST
SOFTWARE
SOFTWARE_CONFIG
```

Because this data structure is not easy to handle, there is a table HARDWARE_INFO which collects the information of different devices:

```
CREATE TABLE  `opsi`.`HARDWARE_INFO` (
  `config_id` int(11) NOT NULL,            //Verweis auf Device Configtabelle
  `host_id` int(11) NOT NULL,              //Verweis auf host-Tabelle
  `hardware_id` int(11) NOT NULL,          //Verweis auf Device Tabelle
  `hardware_class` varchar(50) NOT NULL,   //Device
  `audit_firstseen` timestamp NOT NULL,    //
  `audit_lastseen` timestamp NOT NULL,
  `audit_state` tinyint(4) NOT NULL,       //1=aktuell 0=nicht mehr aktuell
  `internalConnectorType` varchar(60) ,
  `verticalResolution` int(11),
  `totalPhysicalMemory` bigint(20),
  `family` varchar(50) ,
  `vendorId` varchar(4) ,
  `addressWidth` tinyint(4),
  `videoProcessor` varchar(20) ,
  `numberOfFunctionKeys` int(11),
  `maxDataWidth` tinyint(4),
  `memoryType` varchar(20) ,
  `maxSize` int(11),
  `tag` varchar(100) ,
  `voltage` double,
  `slots` tinyint(4),
  `screenWidth` int(11),
  `connectorType` varchar(60) ,
  `maxCapacity` bigint(20),
  `size` bigint(20),
  `formFactor` varchar(10) ,
  `driveLetter` varchar(2) ,
  `capacity` bigint(20),
  `socketDesignation` varchar(100) ,
  `externalConnectorType` varchar(60) ,
  `numberOfButtons` tinyint(4),
  `capabilities` varchar(200) ,
  `port` varchar(20) ,
  `dataWidth` tinyint(4),
  `horizontalResolution` int(11),
  `version` varchar(50) ,
  `maxClockSpeed` bigint(20),
  `location` varchar(50) ,
  `paperSizesSupported` varchar(200) ,
  `deviceType` varchar(10) ,
  `subsystemVendorId` varchar(4) ,
  `adapterRAM` bigint(20),
  `speed` int(11),
  `architecture` varchar(50) ,
  `status` varchar(20) ,
  `freeSpace` bigint(20),
  `product` varchar(100) ,
  `vendor` varchar(50) ,
  `description` varchar(100) ,
  `index` int(11),
  `systemType` varchar(50) ,
  `macAddress` varchar(20) ,
  `installedSize` int(11),
  `driverName` varchar(100) ,
  `subsystemDeviceId` varchar(4) ,
  `internalDesignator` varchar(60) ,
```

```
  `currentUsage` varchar(20) ,
  `extClock` int(11),
  `heads` int(11),
  `autoSense` varchar(20) ,
  `currentClockSpeed` bigint(20),
  `netConnectionStatus` varchar(20) ,
  `partitions` tinyint(4),
  `maxSpeed` int(11),
  `busId` varchar(60) ,
  `name` varchar(100) ,
  `sectors` bigint(20),
  `level` varchar(10) ,
  `serialNumber` varchar(50) ,
  `screenHeight` int(11),
  `startingOffset` bigint(20),
  `externalDesignator` varchar(60) ,
  `filesystem` varchar(50) ,
  `cylinders` int(11),
  `model` varchar(100) ,
  `revision` varchar(4) ,
  `deviceLocator` varchar(100) ,
  `adapterType` varchar(20) ,
  `deviceId` varchar(4) ,
  PRIMARY KEY  (`config_id`,`host_id`,`hardware_class`,`hardware_id`)
)
```

Which field name in the database is corresponding to which reported and localized name in the opsi management interface is defined in a configuration file. Example (/etc/opsi/hwaudit/locales/de_DE):

```
DEVICE_ID.deviceType = Gerätetyp
DEVICE_ID.vendorId = Hersteller-ID
DEVICE_ID.deviceId = Geräte-ID
DEVICE_ID.subsystemVendorId = Subsystem-Hersteller-ID
DEVICE_ID.subsystemDeviceId = Subsystem-Geräte-ID
DEVICE_ID.revision= Revision
BASIC_INFO.name = Name
BASIC_INFO.description = Beschreibung
HARDWARE_DEVICE.vendor = Hersteller
HARDWARE_DEVICE.model = Modell
HARDWARE_DEVICE.serialNumber = Seriennummmer
COMPUTER_SYSTEM = Computer
COMPUTER_SYSTEM.systemType = Typ
COMPUTER_SYSTEM.totalPhysicalMemory = Arbeitsspeicher
BASE_BOARD = Hauptplatine
BASE_BOARD.product = Produkt
BIOS = BIOS
BIOS.version = Version
SYSTEM_SLOT = System-Steckplatz
SYSTEM_SLOT.currentUsage = Verwendung
SYSTEM_SLOT.status = Status
SYSTEM_SLOT.maxDataWidth = Max. Busbreite
PORT_CONNECTOR = Port
PORT_CONNECTOR.connectorType = Attribute
PORT_CONNECTOR.internalDesignator = Interne Bezeichnung
PORT_CONNECTOR.internalConnectorType = Interner Typ
```

```
PORT_CONNECTOR.externalDesignator = Externe Bezeichnung
PORT_CONNECTOR.externalConnectorType = Externer Typ
PROCESSOR = Prozessor
PROCESSOR.architecture = Architektur
PROCESSOR.family = Familie
PROCESSOR.currentClockSpeed = Momentane Taktung
PROCESSOR.maxClockSpeed = Maximale Taktung
PROCESSOR.extClock = Externe Taktung
PROCESSOR.processorId = Prozessor-ID
PROCESSOR.addressWidth = Adress-Bits
PROCESSOR.socketDesignation = Zugehöriger Sockel
PROCESSOR.voltage = Spannung
MEMORY_BANK = Speicher-Bank
MEMORY_BANK.location = Position
MEMORY_BANK.maxCapacity = Maximale Kapazität
MEMORY_BANK.slots = Steckplätze
MEMORY_MODULE = Speicher-Modul
MEMORY_MODULE.deviceLocator = Zugehöriger Sockel
MEMORY_MODULE.capacity = Kapazität
MEMORY_MODULE.formFactor = Bauart
MEMORY_MODULE.speed = Taktung
MEMORY_MODULE.memoryType = Speichertyp
MEMORY_MODULE.dataWidth = Datenbreite
MEMORY_MODULE.tag = Bezeichnung
CACHE_MEMORY = Zwischenspeicher
CACHE_MEMORY.installedSize = Installierte Größe
CACHE_MEMORY.maxSize = Maximale Größe
CACHE_MEMORY.location = Position
CACHE_MEMORY.level = Level
PCI_DEVICE = PCI-Gerät
PCI_DEVICE.busId = Bus-ID
NETWORK_CONTROLLER = Netzwerkkarte
NETWORK_CONTROLLER.adapterType = Adapter-Typ
NETWORK_CONTROLLER.maxSpeed = Maximale Geschwindigkeit
NETWORK_CONTROLLER.macAddress = MAC-Adresse
NETWORK_CONTROLLER.netConnectionStatus = Verbindungsstatus
NETWORK_CONTROLLER.autoSense = auto-sense
AUDIO_CONTROLLER = Audiokarte
IDE_CONTROLLER = IDE-Controller
SCSI_CONTROLLER = SCSI-Controller
FLOPPY_CONTROLLER = Floppy-Controller
USB_CONTROLLER = USB-Controller
1394_CONTROLLER = 1394-Controller
PCMCIA_CONTROLLER = PCMCIA-Controller
VIDEO_CONTROLLER = Grafikkarte
VIDEO_CONTROLLER.videoProcessor = Video-Prozessor
VIDEO_CONTROLLER.adapterRAM = Video-Speicher
DRIVE.size = Größe
FLOPPY_DRIVE = Floppylaufwerk
TAPE_DRIVE = Bandlaufwerk
HARDDISK_DRIVE = Festplatte
HARDDISK_DRIVE.cylinders = Cylinder
HARDDISK_DRIVE.heads = Heads
HARDDISK_DRIVE.sectors = Sektoren
HARDDISK_DRIVE.partitions = Partitionen
DISK_PARTITION = Partition
DISK_PARTITION.size = Größe
```

```
DISK_PARTITION.startingOffset = Start-Offset
DISK_PARTITION.index = Index
DISK_PARTITION.filesystem = Dateisystem
DISK_PARTITION.freeSpace = Freier Speicher
DISK_PARTITION.driveLetter = Laufwerksbuchstabe
OPTICAL_DRIVE = Optisches Laufwerk
OPTICAL_DRIVE.driveLetter = Laufwerksbuchstabe
MONITOR = Monitor
MONITOR.screenHeight = Vertikale Auflösung
MONITOR.screenWidth = Horizontale Auflösung
KEYBOARD = Tastatur
KEYBOARD.numberOfFunctionKeys = Anzahl Funktionstasten
POINTING_DEVICE = Zeigegerät
POINTING_DEVICE.numberOfButtons = Anzahl der Tasten
PRINTER = Drucker
PRINTER.horizontalResolution = Vertikale Auflösung
PRINTER.verticalResolution = Horizontale Auflösung
PRINTER.capabilities = Fähigkeiten
PRINTER.paperSizesSupported = Unterstützte Papierformate
PRINTER.driverName = Name des Treibers
PRINTER.port = Anschluss
```

Examples for queries:

Complete hardware inventory ordered by clients and devices:

```
select
HOST.hostId,HARDWARE_INFO.*
from
HOST,HARDWARE_INFO
where
(HOST.host_id = HARDWARE_INFO.host_id)
ORDER BY
HOST.hostId,HARDWARE_INFO.hardware_class,HARDWARE_INFO.config_id
```

Complete hardware inventory of one client ordered by devices:

```
select
HOST.hostId,HARDWARE_INFO.*
from
HOST,HARDWARE_INFO
where
(HOST.host_id = HARDWARE_INFO.host_id)
and HOST.hostId = 'pcuwb03.uib.local'
ORDER BY
HOST.hostId,HARDWARE_INFO.hardware_class,HARDWARE_INFO.config_id
```

Listing of all harddrives:

```
SELECT * FROM HARDWARE_DEVICE_HARDDISK_DRIVE D
LEFT OUTER JOIN HARDWARE_CONFIG_HARDDISK_DRIVE H ON
D.hardware_id=H.hardware_id ;
```

12.3.2. Initializing the MySQL-Backend

First, the mysql-server has to be installed (if not done yet):

```
apt-get install mysql-server
```

In the next step the administrative password for the mysql-server has to been set:

```
mysqladmin --user=root password linux123
```

Using the script /usr/share/opsi/init-opsi-mysql-db.py you may now initialize the MySQL-Backend.

A example session:

```
svmopside:/usr/share/opsi# ./init-opsi-mysql-db.py

*****************************************************************************
*
*  This tool will create an initial mysql database for use as opsi backend.
*
* The config file /etc/opsi/backendManager.d/21_mysql.conf will be recreated.
*
*                 =>>> Press <CTRL> + <C> to abort <<<=
*
*****************************************************************************
*

 Database host [localhost]:
 Database admin user [root]:
 Database admin password [password]:
 Opsi database name [opsi]:
 Opsi database user [opsi]:
 Opsi database password [opsi]:

Connecting to host 'localhost' as user 'root'
Creating database 'opsi' and user 'opsi'
Testing connection
Connection / credentials ok!
Creating mysql backend config file /etc/opsi/backendManager.d/21_mysql.conf
Creating opsi base
Got duplicate property 'name' of same type 'varchar' but different sizes: 100,
50
Using type varchar(100) for property 'name'
Got duplicate property 'name' of same type 'varchar' but different sizes: 100,
60
Using type varchar(100) for property 'name'
Got duplicate property 'name' of same type 'varchar' but different sizes: 100,
50
Using type varchar(100) for property 'name'
Got duplicate property 'location' of same type 'varchar' but different sizes:
50, 10
Using type varchar(50) for property 'location'
Got duplicate property 'name' of same type 'varchar' but different sizes: 100,
50
Using type varchar(100) for property 'name'
```

At all questions (beside the password) you may accept the defaults by pressing ENTER.

Warnings at the end of the script should be ignored.

In the file /etc/opsi/backendManager.d/30_vars.conf is defined which backend is used for which part of opsi. In order to use the MySQL backend, the inventory of hard- and software has to be assigned to BACKEND_MYSQL in this file, no matter which backend is used otherwise.

```
self.swinventBackend        = BACKEND_MYSQL
self.hwinventBackend        = BACKEND_MYSQL
```

After changing the backend configurationthe opsiconfd must be restarted:

```
/etc/init.d/opsiconfd restart
```

12.4. Conversion between different backends

The command **opsi-convert** converts the opsi configuration files from one backend to another. The target or the source can be assigned in different ways:

- Backend name
  A backend on the current server can be addressed with just the backend name. The command '**opsi-convert File File31**' converts the data base of the current server from File-Backend to File31-Backend.

- Service address
  Providing a full qualified service address allows access to a remote servers data base (after passing the users password). The service address looks like:
  https://<username>@<ipadresse>:4447/rpc
  The conversion command looks like that:
  **opsi-convert -s -l /tmp/log https://uib@192.168.2.162:4447/rpc **
  https://opsi@192.168.2.42:4447/rpc

- Configuration directories
  With a declaration of a configuration directory for the specified backend manager configuration source or target can be described in detail.

12.5. Boot files

'/tftpboot/linux' contains the boot files needed for the system start with the PXE-Bootproms.

12.6. Securing the shares with encrypted passwords

The installation software 'opsi preLoginLoader' accesses the shares provided by the depot server in order to install software and to write configuration information and log files. This is done with the privileges of the system user 'pcpatch'. Securing these shares and therefore the authentication data of 'pcpatch' is important for two reasons:

- general system security and data integrity
- meet the license agreements of special software packets

To give the client task 'preLoginLoader' access to authentication data, the server task 'reInstallationManager' creates a specific key when preparing a client re-installation request. This key is stored in the file '/etc/pckeys' and is passed to the PC with the reinstallation request. The client PC will store this key in the local file 'c:\opsi\cfg\locked.cfg' during system installation (access rights limited to the administrators). Also, on the server, the file '/etc/pckeys' is only accessible by user root. This way every PC has got an unique key only known to the client itself and the depot server, not accessible by client standard users. The key is used to encrypt the password of the user 'pcpatch'. The encrypted password will be transferred to the client at boot time via webservice. Hence the servers 'pcpatch' password can be changed any time. The new encrypted password will be sent to every client at the next reboot.

13. Adapting the opsi preloginLoader to your Corporate Identity (CI)

Since opsi-winst version 4.6 it is possible to manipulate the graphical skin of this program by changing the file bg.png in the subdirectory 'winstskin'. After any change at this file in the directory opt/pcbin/install/opsi-winst/files/opsi-winst/winstskin you should execute the command touch /opt/pcbin/install/opsi-winst/files/opsi-winst/winst32.exe which canges the timestamp of this file. As result of this operation the preloginloader will see a 'changed' winst at the next startup and copy the winst and all it files (including bg.png) to the client.

If you want to cange the outfit of the preloginloader agent itself we recommend to use the prloginloader version 3.4. In this Version it is possible to manipulate the skin files notifier\action.bmp and notifier\event.bmp according to your needs.

14. Overview: A PC boots from the network

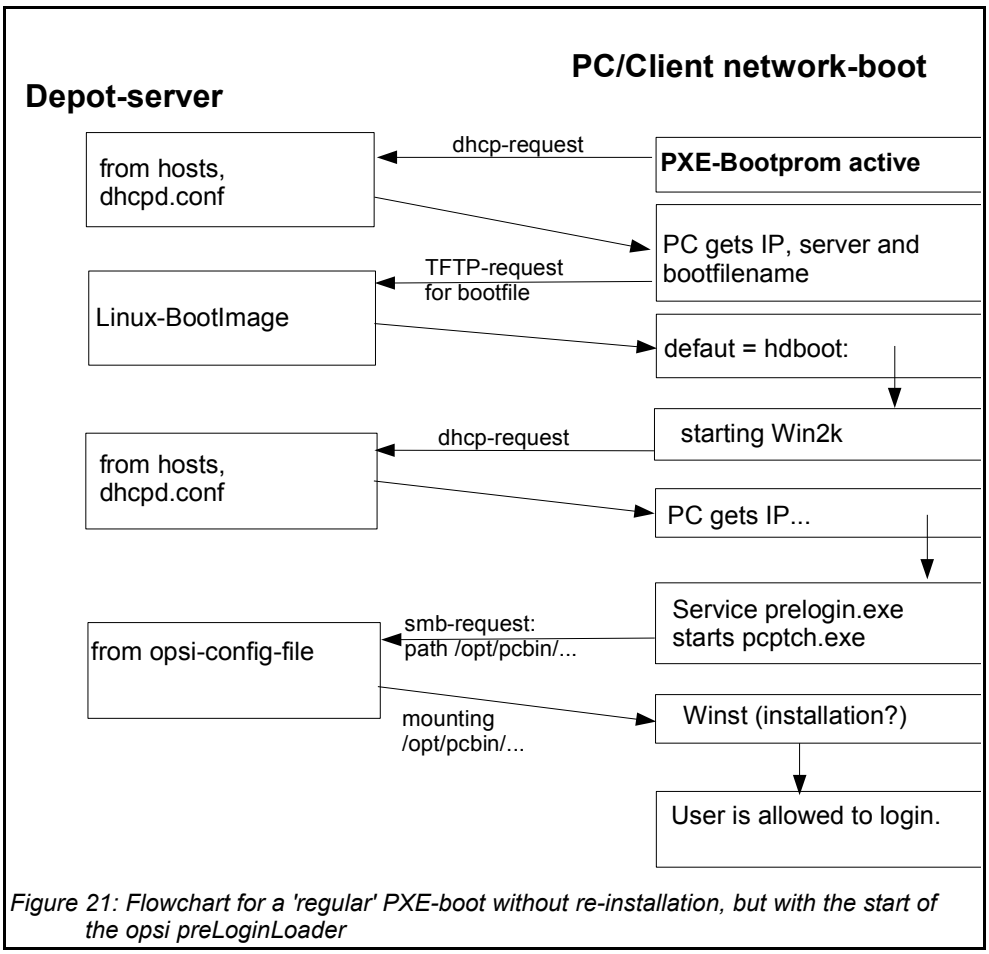


Figure 21: Flowchart for a 'regular' PXE-boot without re-installation, but with the start of the opsi preLoginLoader

Again the first action is the bootprom sending its DHCP-request to the network.

In case of a boot without an OS (re)installation the bootimage cannot get a PC-specific '01-<MAC>'-file, instead the default file is loaded and initiates a local boot. During the startup of the OS, the OS will again request an IP and the usual network configuration information from the DHCP-server.

15. Important files on the depot servers

15.1. Configuration files

15.1.1. Configuration files in /etc

15.1.1.1. /etc/hosts

The hosts file stores all IP addresses and IP names known to the network. The IP addresses and names of all clients have to be entered here. The names have to be 'full qualified', including the domain name. There might be aliases (additional names) and comments (starting with '#').

Example:

```
192.168.2.104  laptop1.uib.local  laptop1
192.168.2.106  laptop2.uib.local  laptop2     # you can enter comments here
192.168.2.153  desktop1.uib.local desktop1
192.168.2.178  test_pc1.uib.local test_pc1    # Test-PC PXE-bootprom
```

15.1.1.2. /etc/group

The required opsi groups are 'pcpatch' and 'opsidamin'. All users who are administrating opsi packets need to be member of the 'pcpatch' group. Membership of the group 'opsiadmin' allows users to connect to the opsi web service (for instance using the opsi--Configed or the applet).

15.1.1.3. /etc/opsi/pckeys

In this file the keys for the re-Installation manager, specified for each computer, are stored.

Example:

```
laptop1.uib.local:fdc2493ace4b372fd39dbba3fcd62182
laptop2:c397c280fc2d3db81d39b4a4329b5f65
desktop1.uib.local:61149ef590469f765a1be6cfbacbf491
```

15.1.1.4. /etc/opsi/passwd

Here the passwords encrypted with the server key of the server (e.g. for pcpatch) are kept.

15.1.1.5. /etc/opsi/backendManager.conf

Deprecated since opsi 3.1 and replaced by '/etc/opsi/backendManager.conf.d/*',

Configuration file for the opsiconfd specifying which backend (File/LDAP) will be used, where the data storage is and what commands are bound to what actions.

15.1.1.6. /etc/opsi/backendManager.conf/*

Since opsi version 3.1

Configuration files for the 'opsiconfd' service defining
- which backend (File/LDAP) to use,
- where to store the data files,
- which commands are bound to what action,
- the list of available service requests.

The *.conf files of this directory in alphabetic order will be combined to one single file at run time to build the backendManager.conf. Also custom specific files can be included to override the default settings (without losing this information at the next update).

15.1.1.7. /etc/opsi/hwaudit/*

Since opsi V3.2

Here the configuration files for the hardware inventory are to be found. The directory 'locales' holds the language specifications. The file 'opsihwaudit.conf' specifies the mapping of WMI classes to the opsi data management.

15.1.1.8. /etc/opsi/opsiconfd.conf

Since opsi V3

Configuration file for the 'opsiconfd' service including configurations like ports, interfaces, logging.

15.1.1.9. /etc/opsi/opsiconfd.pem

Since opsi version 3.0

Configuration file for the 'opsiconfd' holding the ssl certificate.

15.1.1.10. /etc/opsi/opsipxeconfd.conf

Configuration file for the 'opsipxeconfd' in charge for writing the startup files for the Linux-bootimage. You can configure directories, defaults and log level here.

15.1.1.11. /etc/opsi/version

Holds the version number of the installed opsi.

15.1.1.12. /etc/init.d/

Start and stop scripts for

- opsi-atftpd

v3 opsiconfd

v3.1 opsipxeconfd

15.2. Boot files

15.2.1. Boot files in /tftpboot/linux

15.2.1.1. pxelinux.0

Bootfile which will be loaded first by the PXE-bootprom.

15.2.1.2. install und miniroot.gz

Installation bootimage which will be loaded by the client (per tftp) during a re-installation.

15.2.2. Boot files in /tftpboot/linux/pxelinux.cfg

15.2.2.1. 01-<MAC address> or <IP-NUMBER-in-Hex>

Files named by the clients hardware address (prefix 01-) are stored on the depot server as client-specific boot files. Usually they are named pipes created by the re-InstallationManager as to initiate the (re)installation of clients.

15.2.2.2. default

The file 'default' is loaded if no client-specific file is found. This initiates a local boot.

15.2.2.3. install

Information for the boot of the install boot image which will be used by the opsi-re-installationManager to create the named pipe.

15.3. Files of the File-Backend

Attention: opsi can be configured in many ways. The file locations as documented here are the opsi defaults. The actual locations are to be found in the /etc/opsi/backendManager.conf.d/* configuration files.

15.3.1. File3.1-Backend

15.3.1.1. Overview

The files of the 'File31 backend' are in '/var/lib/opsi', which is the home directory of the opsiconf-daemons. The following schema gives an overview of the directory structure.

15. Important files on the depot servers

```
/var/lib/opsi┐
             ├─depot/                    (for future use: depotshare)
             ├─log/                      (for future use: logshare)
             ├─utils/                    (for future use: utilsshare)
             ├─config/┐                  configshare
                      ├─clientgroups.ini  Client groups
                      ├─global.ini        network and additional config
                      ├─clients/          (<pcname.ini> files)
                      ├─templates/        (templates for <pcname.ini>
                      ├─depots/┐
                               ├─<depotid>/┐
                                           ├─depot.ini
                                           ├─products/┐
                                                      ├─localboot/┐
                                                      │  (product control
                                                      │   files)
                                                      └─netboot/┐
                                                         (product control
                                                         files)
```

- Logging and hard- and software inventory
  The hardware information sampled by the product 'hwaudit' or the bootimage are saved as '<configshare>/pclog/<pcname>.hw'.
  The software information sent from the product 'swaudit' is saved as '<configshare>/pclog/<pcname>.sw'.

15.3.1.2. Configuration files in '/var/lib/opsi/config'

15.3.1.2.1. clientgroups.ini

This file holds information on the client-groups.

```
[groupname]
membername
membername
(....)
```

Example

```
[group 3]
pca26
pca39
pcmeyer
```

15.3.1.2.2. global.ini

This file contains the default settings of the sections **[networkconfig]** and **[generalconfig]** for the client configuration. Client specific values from '<pcname>.ini' will override these default values. The inner structure of these sections is the same as described in the next chapter for '<pcname>.ini'.

15.3.1.3. Configuration files in /var/lib/opsi/config/clients

15.3.1.3.1. <pcname>.ini

In these files the client specific configuration is set. This information will be combined with the 'global.ini' values whereas the settings from '<pcname>.ini' overrides the 'global.ini' setting.

These files can have the following sections:

15.3.1.3.1.1. [generalconfig]

In this section are the general client entries. Values from this section will be transferred by the service request 'getGeneralConfig_hash' and the bootimage to patch the configuration files entries.

Example:

```
pcptchbitmap1 = wInst1.bmp
pcptchbitmap2 = wInst2.bmp
pcptchlabel1 = opsi
pcptchlabel2 = uib gmbh
```

Icons and labeling of the pcpatch.exe 'netmount' window

```
SecsUntilConnectionTimeOut = 120
```

Timeout of pcptch.exe ('netmount' window) – if no server connection is available

```
button_stopnetworking=immediate
```

The 'netmount' window should present the 'cancel'-button right from the start

```
test = 123
```

any user defined keys

```
os = winxppro
```

Default value for operating system installation

15.3.1.3.1.2. [networkconfig]

```
depoturl=smb://<smbhost>/<sharename>/<path>
configurl=smb://<smbhost>/<sharename>/<path>
utilsurl=smb://<smbhost>/<sharename>/<path>
```

The URL consists of three parts:

1. Protocol: Currently only the 'smb' protocol is supported.
2. Share name (for instance '\\laptop\opt_pcbin'): This share will be mounted. In case of a drive letter given further down in this file, the share is mounted as this drive.
3. The path where the installation software is stored.

```
depotdrive=<drive letter the depoturl will be mounted as>
```

Example: P: (including the colon)

```
configdrive=<drive letter the configurl will be mounted as>
```

Example: P: (including the colon)

```
utilsdrive=<drive letter the utilsurl will be mounted as>
```

Example: P: (including the colon)

```
nextbootservertype = service
```

The client can work with the opsi-service or with direct data access ('classic' mode). Classic mode is available with the 'File' backend only, but not with the 'File31' or 'LDAP' backend. The client will retrieve that value and save it as '**[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\pcptch] opsiServerTyp**'.

```
nextbootserviceurl = https://192.168.1.14:4447
```

This is the URL the client connects to the opsi service running on the server. Attention: If there is a name and not an IP-number, the name must be resolvable by the client. The value will be retrieved by the client and saved as '**[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\pcptch] opsiServiceUrl**'.

```
windomain = dplaptop
```

This is the name of the Samba(Windows)-domain

15.3.1.3.1.3. [localboot_product_states]

Replaces the deprecated section [products-installed] and looks like:

```
<productid> = <installation state> : <required action>
```

e.g.

```
firefox = installed:setup
```

15.3.1.3.1.4. [netboot_product_states]

```
<productid> = <installation state> : <required action>
```

e.g.

```
winxppro = installed:none
```

15.3.1.3.1.5. [<product>-state]

This section holds information on every software packet installed on the client including time stamp of installation.

```
laststatechange = <timestamp>
packageversion = <value>
productversion = <value
```

e.g.

```
laststatechange = 20070525105058
packageversion = 1
productversion = 2.0.0.3
```

15.3.1.3.1.6. [<product>-install]

```
product property = value
```

e.g.

```
viewer = off
```

15.3.1.3.1.7. [info]

The client information from the opsi-configed will be saved to the 'info' section. Also will be recorded here the last time the client connected the 'opsiconfd' service.

```
[info]
notes =
description = detlef
lastseen = 20070105090525
```

15.3.1.4. Configuration files in /var/lib/opsi/config/templates

In this directory are the template files like 'pcproto.ini', which is the standard template for creating a new <pcname>.ini file. It has the same internal structure as the <pcname>.ini file.

15.3.1.5. Configuration files in /var/lib/opsi/config/depots/<depotid>

In this place is the file 'depot.ini', which is the configuration file of the opsi depot (where on the server the depot is located and how to connect it).

```
[depotShare]
urlForClient = smb://dplaptop/opt_pcbin/install
urlForConfigServer = file:///opt/pcbin/install

[depotServer]
operatingSystem = Linux
```

15.3.1.6. Product control files in /var/lib/opsi/config/depots/<depotid>/products

This directory contains the subdirectories 'localboot' and 'netboot', where the control-files of the respective products are located. The subdirectories contain the product meta data, which is the product name, properties, default values and dependencies.

The control files are the kind of control files, that are generated by creating new opsi-products in the directory '<product name>/OPSI/control'.

The control files have the following sections:

- Section [Package]
  Description of the package version and whether this is an incremental package.
- Section [Product]
  Description of the product
- Optional section(s) [ProductProperty]
  Description of variable product properties
- Optional section(s) [ProductDependency]
  Description of product dependencies

Example:

```
[Package]
version: 1
depends:
incremental: False

[Product]
type: localboot
id: thunderbird
name: Mozilla Thunderbird
description: Mail client of Mozilla.org
advice:
version: 2.0.0.4
priority: 0
licenseRequired: False
productClasses: Mailclient
setupScript: thunderbird.ins
uninstallScript:
updateScript:
alwaysScript:
onceScript:

[ProductProperty]
name: enigmail
description: Install encryption plugin for GnuPG
values: on, off
default: off

[ProductDependency]
action: setup
requiredProduct: mshotfix
requiredStatus: installed
requirementType: before
```

- [Package]-'Version' is for different package versions from the same product version. This helps to distinguish packages build from the same product version but with different wInst-script for instance.
- [Package]-'depends' refers to the base package of an incremental package.
- [Package]-'Incremental' specifies whether this is an incremental package.
- [Product]-'type' marks the product type as localboot or netboot.
- [Product]-'Id' is the general name of that product (like 'firefox'), independent from the product version (with opsi 2 this is called the 'product name').
- [Product]-'name' is the full name of the product.

- [Product]-'Description' is an additional description for the product as shown in the opsi-Configeditor as 'Description'.
- [Product]-'Advice' is an additional hint for handling the product (caveats etc.) as to be shown in the opsi-Configeditor as 'Note'.
- [Product]-'version' is the version of the original software.
- [Product]-'Priority' is for future use (regarding the installation order).
- [Product]-'class' is for future use.
- [ProductProperty]-'name': Name of a properties.
- [ProductProperty]-'description': Description of the properties (shown as tool tip in opsiconfiged).
- [ProductProperty]-'values' : List of allowed values. If empty, the value is free editable.
- [ProductProperty]-'default' : Default value of the property.
- [ProductDependency]-'Action' : To which product action this dependency entry belongs (setup, deinstall ...).
- [ProductDependency]-'Requiredproduct': Product ID of the product a dependency exists.
- [ProductDependency]-'Required action': The required action of the product, which the dependency entry refers to. Actions could be setup, deinstall, update...
- [ProductDependency]-'Required installation status': The required status of the product, which the dependency entry refers to. Typically this is 'installed', which results in setting this dependency product to setup, if it isn't installed on the client yet.
- [ProductDependency]-'Requirement type': this is regarding the installation order. If the product, which the dependency entry refers to, has to be installed before the actual product installation starts, the 'Requirement type' must be 'before'. If

the dependency product has to be (re-)installed after the actual product, the 'Requirement type' is set to 'after'. If there is no entry, the installation order is of no relevance.

15.4. Files of the LDAP-backend

The opsi-LDAP schema is located in the directory **`/etc/ldap/schema/opsi.schema.`**

15.5. Opsi programs and libraries

15.5.1. Python library

The opsi python modules are located at:

```
/usr/lib/python2.3/site-packets/OPSI/
```

or

```
/usr/share/python-support/python-opsi/OPSI
```

15.5.2. Programs in /usr/sbin

v3 opsiconfd
opsi configuration daemon

v3.1 opsipxeconfd
opsi daemon to administrate the files required for the PXE-boot of the clients.

15.5.3. Programs in /usr/bin

- opsi-admin
  Starts the command line interface for the opsi python library
- opsi-configed
  Command to start the opsi-management interface
- opsi-convert
  Script for converting between different backends.

- opsideinst
  (deprecated - replaced by opsi-package-manager) Script for deleting products
- opsiinst (opsiinstv2)
  (deprecated - replaced by opsi-package-manager) Script to unpack and install opsi packets on the server
- opsi-makeproductfile (opsi-makeproductfilev2)
  Script for packing the opsi-packet (opsiV2 compatible Version)
- opsi-newprod
  Script for creating a new opsi product
- opsi-package-manager
  Script to unpack, install, remove, list opsi packets on one ore more servers.
- makeproductfile (makeproductfilev2) (deprecated)
  Replaced by opsi-makeproductfile
  Script for packing the opsi-packet (opsiV2 compatible Version)
- newprod (deprecated)
  Replaced by opsi-newprod
  Script for creating a new opsi product
- sysbackup
  System backup (to tape or disc)
- winipatch
  Script for patching INI-files

15.6. opsi-log files

15.6.1. /var/log

The opsi reInstallationManager logs to '/var/log/syslog'.

15.6.2. /var/log/opsi/opsiconfd

In this directory are the log-files of the opsiconfd and the clients. The client log files will be named 'log.<IP-number>' and (if available) a symbolic link named 'log.<IP-name>' to 'log.<IP-number>' is created.

15.6.3. /var/log/opsi/bootimage

In this directory are the log-files of the opsi-bootimage. These log files will be named 'log.<IP-number> If the boot image couldn't connect the webservice, the logs are written to '/tmp/log' at the bootimage.

15.6.4. /var/log/opsi/opsipxeconfd

This is the log file of the opsipxeconfd, that administrates the tftp files for the PXE boot of the clients.

15.6.5. Software installation (c:\tmp)

The logging of the opsi preloginloader service 'prelogin.exe' is managed by the registry entry '[HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\preloginloader] DebugOutput'. Usually it is set to '0', which means 'no logging'. For debugging it can be set to 1 (some logging), 2, 3 or 4 (verbose logging). The logs are shown in the Windows event viewer in the opsi section.

The netmount program 'pcptch.exe' logs to 'c:\tmp\logonlog.txt'.

The opsi-wInst writes a detailed log of its current activities to 'c:\tmp\instlog.txt'. This will be overwritten at the next start. The cumulative error log file is 'c:\tmp\instlog.err' and can be configured to under the name instlog.txt in c:\tmp. The error log can be configured to be written to the config share as '<configshare>/pclog/<pcname>.err' or to transfer the error logs per syslog protocol to a log server.

16. Registry entries

16.1. Registry entries for the opsi-preLoginLoader

16.1.1. opsi.org/general

The following entries will be found in the registry at [HKLM/Software/opsi.org/general].

```
configlocal = <0/1> (dword)
```

If 'configlocal=0' all the following keys are updated at every boot with information from the sysconf files. The sysconf files are retrieved by the client at every boot from the server via tftp. So the client needs to know the address of the tftpserver:

```
tftpserver = <server to get the configuration files from>
```

16.1.2. opsi.org/shareinfo

The following registry entries are stored in [HKLM/Software/opsi.org/shareinfo]:

```
user = <user to mount the shares>
```

Example for user: pcpatch

```
pcpatchpass = <blowfish encrypted password for user pcpatch>
depoturl = <URL for installation packets>
; depoturl pattern: <protocol>:\\<server>\<share>\<dir>
```

Example for depoturl: smb:\\laptop\opt_pcbin\install

The URL consists of three parts:

1. Protocol (smb): Currently only smb is supported.
2. Share (\\laptop\opt_pcbin): This is the share to be mounted. If a drive letter is given for this share the share is mounted to this drive letter.
3. Directory in which the software packets are stored.

```
Configurl = <URL to the configuration files>
; the configuration files are the <pcname>.ini files
; configurl pattern: <protocol>:\\<server>\<share>\<dir>
```

Example for configurl: smb:\\laptop\opt_pcbin\pcpatch

Description: (same structure as depoturl)

```
utilsurl = <URL to the utils directory>
; the utils directory contains the client opsi utilities
; like Winst.exe
; utilsurl pattern: <protocol>:\\<server>\<share>\<dir>
```

Example: smb:\\laptop\opt_pcbin\utils

Description: (same as depoturl)

```
depotdrive = <drive letter the depoturl will be mounted to>
```

Example: P: (including the colon)

```
configdrive = <drive letter the configurl will be mounted to>
```

Example: P: (including the colon)

```
utilsdrive = <drive letter the utilsurl will be mounted to>
```

Example: P: (including the colon)

Configuration values for 'pcptch.exe' in [HKLM/Software/opsi.org/pcptch]

- mountdrive (DWORD) 0=false, 1=true (default 1)
- label1 (String) caption for first image (if empty defaults to "PC-Server-Integration")
- label2 (String) caption for second image (if empty defaults to “uib”)
- Bitmap1 (String) is the name of the first image (BMP file, relative to the path of pcptch.exe, default is 'wInst1.bmp')
- Bitmap2, same as bitmap1for the second image (default is 'wInst2.bmp')

16.1.3. opsi.org/preloginloader

The registry key [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\preloginloader] has the following values:

"PcPCallMode"=dword:00000001 (deprecated)

"DebugOutput"=dword:00000004
- Eventlog: 0=errors only >=4 = verbose

"RebootOnBootmodeReins"=dword:00000001
- Reboot if bootmode=REINS

"RebootOnServicePackChange"=dword:00000001
- reboot if servicepack changed

"WaitForPcpExit"=dword:00000000 (deprecated)

"RemoveMsginaOnDeinst"=dword:00000001
- on uninstall restore the default login handler

"UtilsDir"="C:\\opsi\\utils"
- path to the preloginloader files

"PcptchExe"="C:\\opsi\\utils\\pcptch.exe"
- task to start (default is 'pcptch.exe')

"WinstRegKey"="SOFTWARE\\Hupsi\\wInst"
- where to look for wInst registry reboot requests

"LoginBlockerStart"=dword:00000001
- pgina waits for READY from the named pipe
(if set to 0, the user is allowed to logon during software installation)

"LoginBlockerTimeout"=dword:00000300
- Timeout in minutes for 'wait for ready' (then allow login)

"LoginBlockerTimeoutConnect"=dword:00000005
- Timeout in minutes for pipe-connect

opsi.org/pcptch

Key [HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\pcptch]

“SecsUntilConnectionTimeOut“=“10“
- wait 10 seconds for a network connection, otherwise continue
- value = 0 -> deactivated

If the entry “button_stopnetworking” is set to “immediate”, the button to cancel the network connection will be shown immediately. Laptops (which are most of the time offline) should be configured as “immediate”, so the installation task (which is trying to connect the installation share) can be stopped immediately.

- opsiServerType
  Defines whether 'pcptch.exe' (in opsi 2 mode) should operate file based or connect to the opsi service.
  Possible values are:
  classic -> opsi 2 mode
  service -> opsi 3 mode

- opsiServiceURL
  The URL to connect the opsi service
  e.g. https://bonifax.uib.local:4447

- repeatServiceConnectNo
  Number of retries for connecting the service (default 3).

16.2. Registry-entries for opsi-wlnst

16.2.1. Controlling the logging via syslog protocol

The relevant registry section is [HKLM\Software\opsi.org\syslogd]

the value of 'RemoteErrorLogging' (DWORD) is evaluated:
RemoteErrorLogging = (0=trel_none, 1=trel_filesystem, 2=trel_syslog);

If logging is set to syslog protocol ("remoteerrorlogging"=dword:00000002), the string variable 'sysloghost' gives the IP-name of the LogHost.

The DWORD variable 'syslogfacility' defines the source of the syslog messages (default is ID_SYSLOG_FACILITY_LOCAL0).

The logging source can be:

```
ID_SYSLOG_FACILITY_KERNEL       = 0;  // kernel messages
ID_SYSLOG_FACILITY_USER         = 1;  // user level messages
```

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

17. Supplement: Update of a opsiserver

17.1. Update 3.3.1 to 3.4

17.1.1. Documentation

Please read the opsi 3.4 changes documentation in the opsi-manual.

17.1.2. Backup

It is always a good idea to make a backup before relevant changes to the system

At this release there are major changes at the MySQL-Backend. So a backup of the existing data base is strongly recommended.

This is a possible command to do this job:

```
/etc/init.d/mysql stop
cp -a /var/lib/mysql /var/lib/mysql.backup
/etc/init.d/mysql start
```

17.1.3. Debian / Ubuntu

17.1.3.1. Register of the opsi 3.4 repository

In order to avoid that an update to 3.4 happens accidentally, the debian package for opsi 3.4 is in a specific repository. Delete in /etc/apt/sources.list the entry:
deb http://download.uib.de/debian etch opsi3.3.1
and put in:

For Debian Etch, Ubuntu Dapper/Edgy/Feisty (i386/amd64):

deb http://download.uib.de/debian lenny opsi3.4

For Debian Lenny, Ubuntu Hardy (i386/amd64):

deb http://download.uib.de/debian lenny opsi3.4

Remark: Ubuntu Jaunty isn't supported yet.

Execute **aptitude update**.

17.1.3.2. Put in the opsi debian packages

Put in the packages with following order:
aptitude safe-upgrade

If you be asked on upgrading which version of a configuration file you wish to apply you should choose the newest version. If not you should know exactly what you do e.g. you don't choose the newest version because you want an other as the default File31-Backend.

17.1.4. Suse

Important: Please do not upgrade from opsi3.3 to opsi3.4 directly. If you are using opsi3.3 please upgrade to opsi3.3.1 first or contact uib-opsi-support.

In order to update execute the following commands:

```
zypper rr opsi3.3.1
zypper ar http://download.uib.de/suse/opsi3.4 opsi3.4
zypper update
```

17.1.5. Checking the backend configuration

In the file /etc/opsi/backendManager.d/30_vars.conf is defined which backend manage of opsi be used (**BACKEND_FILE31, BACKEND_MYSQL, BACKEND_LDAP**).

The default backend is **BACKEND_FILE31.**

The backend **BACKEND_FILE** is deprecated - it is not supported anymore**.**

In the entry **clientManagingBackend** may be controlled if opsi also assume the local DHCP configuration. This is sensible if the DHCP-server of the opsiserver will be used (default). The accordant entry is:

```
self.clientManagingBackend = [ BACKEND_DHCPD, BACKEND_FILE31 ]
```

If the local DHCP isn't used also the **BACKEND_DHCPD** not required:

```
self.clientManagingBackend = BACKEND_FILE31
```

For the hard- and software inventory you have since opsi 3.3 two possibilities: BACKEND_FILE31 or BACKEND_MYSQL. One of these you have to enter independent which backend is used normally:

```
self.swinventBackend = BACKEND_FILE31
self.hwinventBackend = BACKEND_FILE31
```

The license management module only works with the MYSQL-Backend and need also this Backend for harware and software inventory. In order to use the license management insert:

```
self.swinventBackend        = BACKEND_MYSQL
self.hwinventBackend        = BACKEND_MYSQL
self.licenseBackend         = BACKEND_MYSQL
```

For the logging there is since opsi 3.3 a own Backend: BACKEND_FILE31. These you have to enter independent which backend is used normally:

```
self.loggingBackend = BACKEND_FILE31
```

After adapting the backend configuration the 'opsiconfd' has to be restarted:

```
/etc/init.d/opsiconfd restart
```

17.1.6. MySQL Inventory Backend

If you use the MySql-Backend for inventory and license management you should call

```
/usr/share/opsi/init-opsi-mysql-db.py
```

Perhaps you need to install the mysql-server before you can run this script:

```
aptitude install mysql-server
```

17.1.7. Download of the new opsi products

Fetch the actual necessary opsi packages:

```
cd /home/opsiproducts
mkdir opsi34
cd opsi34

wget -r -l1 -nc -nd -A '*.opsi' http://download.uib.de/opsi3.4/produkte/essential
```

17.1.8. Import of the new opsi products

The downloaded packages has to be installed on the server to be available for the clients. The interactive installation of an opsi package happen with the aid of the order:

```
opsi-package-manager -i <package file name>
```

The following order install the downloaded packages successive:

```
opsi-package-manager -i *.opsi
```

17.1.9. Install and check the activation file

The use of non free components of the opsi installation is controlled by the activation file **`/etc/opsi/modules`** .(See also Chapter Fehler: Referenz nicht gefunden Fehler: Referenz nicht gefunden at page Fehler: Referenz nicht gefunden in this manual).

While the Release Candidate phase you will find a activation file which is valid until 31.8.2009 at http://download.uib.de/opsi3.4/modules . You should copy these file to /etc/opsi using the root account. You may do this with:

```
cd /etc/opsi
wget http://download.uib.de/opsi3.4/modules
```

You may check your activation state with:

```
opsi-admin -d method getOpsiInformation_hash
```

With this activation file you have the possibility to test our license management as well as our Vista / Win7 Support with the new opsiclientd from the prelogloader 3.4. If you are testing the opsiclientd but don't want to buy it, so please remember to change all clients before the evaluation time ends to the legacy prelogin mode. Which mode of the

preloginloader should be installed is controlled by the product property 'client_servicetype' and the file **opt/pcbin/install/preloginloader/files/opsi/cfg/config.ini** with the entry:

```
[installation]
;client_servicetype=prelogin
client_servicetype=opsiclientd
```

You may check the server default for the product property 'client_servicetype' with:

```
opsi-admin -d method getProductProperties_hash preloginloader
```

You may set the server default of product property 'client_servicetype' to 'prelogin' with:

```
opsi-admin -d method setProductProperty preloginloader "client_servicetype" "prelogin"
```

If you have no activation for 'vista', you should work with the 'prelogin' mode.The opsiclientd will not work without the 'vista' activation.

17.1.10. Final 'check' and rollout of the new preloginloader to the clients

All parts of a opsi release are designed to work together. It is no good idea to try running opsi with packages that are mixed from different releases. So you should make a final check if all your packages (Linux-Packages as well as opsi packages) have at least the release version that was published in the release mail on forum.opsi.org or in the announce mailing list.

To avoid running in mixed environments you should rollout the new preloginloader to all your clients soon. If you forget this task, you will perhaps find your self in the situation, that the clients can't connect to the server any more after the next server upgrade.

17.2. Update 3.3 to 3.3.1

17.2.1. Documentation

Please read the opsi 3.3.1 changes documentation in the opsi-manual.

17.2.2. Backup

It is always a good idea to make a backup before relevant changes to the system

If you already have a gina.dll installed which is different from the original msgina (e.g. Novells nwgina) and chained this gina with the opsi-pgina, so note that the Registry entrys of the opsi pgina now located under HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\preloginloader

Regarding the major changes in the structure of the OS-Installation products like winxppro, a backup of the directories of these products below /opt/pcbin/install is strongly recommended.

This is a possible command to do this job:

```
cd /opt/pcbin/install
tar cvf /home/opsiproducts/winxppro-pre3331.tar winxppro
```

17.2.3. Debian / Ubuntu

17.2.3.1. Register of the opsi 3.3.1 repository

In order to avoid that an update to 3.3.1 happens accidentally, the debian package for opsi 3.3.1 is in a specific repository. Delete in /etc/apt/sources.list the entry:
deb http://download.uib.de/debian etch opsi3.3
and put in:

For Debian Etch, Ubuntu Dapper/Edgy/Feisty (i386/amd64):

deb http://download.uib.de/debian etch opsi3.3.1

For Debian Lenny, Ubuntu Hardy (i386/amd64):

deb http://download.uib.de/debian lenny opsi3.3.1

Execute **apt-get update**.

17.2.3.2. Put in the opsi debian packages

Put in the packages with following order:
apt-get upgrade

If you be asked on upgrading which version of a configuration file you wish to apply you should choose the newest version. If not you should know exactly what you do e.g. you don't choose the newest version because you want an other as the default File31-Backend.

17.2.4. Suse

In order to update execute the following commands:

```
zypper rr opsi3.3
zypper ar http://download.uib.de/suse/opsi3.3.1  opsi3.3.1
mkdir /tmp/opsi3.3.1
cd /tmp/opsi3.3.1
wget -Arpm -nH -nd -np -r http://download.uib.de/suse/opsi3.3.1/RPMS/noarch
rm opsi-depotserver* opsi-atftp*
zypper install python-twisted-web python-twisted-conch
rpm -U --nopostun *.rpm
rcopsiconfd  restart
rcopsipxeconfd  restart
```

17.2.5. Checking the backend configuration

In the file /etc/opsi/backendManager.d/30_vars.conf is defined which backend manage of opsi be used (**BACKEND_FILE31, BACKEND_FILE, BACKEND_LDAP**).

The default backend is **BACKEND_FILE31.**

The backend **BACKEND_FILE** is deprecated - you should change to **BACKEND_FILE31.**

In the entry **clientManagingBackend** may be controlled if opsi also assume the local DHCP configuration. This is sensible if the DHCP-server of the opsiserver will be used (default). The accordant entry is:

```
self.clientManagingBackend = [ BACKEND_DHCPD, BACKEND_FILE31 ]
```

If the local DHCP isn't used also the **BACKEND_DHCPD** not required:

```
self.clientManagingBackend = BACKEND_FILE31
```

For the hard- and software inventory you have since opsi 3.3 two possibilities: BACKEND_FILE31 or BACKEND_MYSQL. One of these you have to enter independent which backend is used normally:

```
self.swinventBackend = BACKEND_FILE31
self.hwinventBackend = BACKEND_FILE31
```

For the logging there is since opsi 3.3 a own Backend: BACKEND_FILE31. These you have to enter independent which backend is used normally:

```
self.loggingBackend = BACKEND_FILE31
```

After adapting the backend configuration the 'opsi-confd' has to be restarted:

```
/etc/init.d/opsiconfd restart
```

17.2.6. MySQL Inventory Backend

If you use the MySql-Backend for inventory you should call

```
/usr/share/opsi/init-opsi-mysql-db.py
```

in order to reconfigure your database according to the new hwaudit.conf.

17.2.7. Download of the new opsi products (all users)

Fetch the actual necessary opsi packages:

```
cd /home/opsiproducts
mkdir opsi331
cd opsi331

wget -r -l1 -nc -nd -A '*.opsi' http://download.uib.de/opsi3.3.1/produkte/essential
```

17.2.8. Download of the new opsi products (opsi-vista support customers only)

This chapter concerns only to customers of the opsi support for vista and preloginloader 3.4 which have an account to the opsi vista repository.

Fetch the actual necessary opsi packages from http://download.uib.de/abo/vista/opsi3.3.1 using your account user name and password:

```
cd /home/opsiproducts/opsi331
#the next command is one line
wget -r -l1 -nc -nd --http-user=<user> --http-passwd=<password> -A '*.opsi'
http://download.uib.de/abo/vista/opsi3.3.1
#the same comman again not readable but for cut&paste
wget -r -l1 -nc -nd --http-user=<user> --http-passwd=<password> -A '*.opsi' http://download.uib.de/abo/vista/opsi3.3.1
```

The new opsi-preloginloader 3.4-x replaces the legacy preloginvista product which should be deleted from the server with the following command:

```
opsi-package-manager -r preloginvista
```

The new preloginloader 3.4 replaces also the preloginloader 3.3.1 which have been downloaded before from the general repository. So please delete this package.

The new preloginloader 3.4 is described in the opsi-vista installation manual: http://download.uib.de/opsi3.3.1/doku/opsi_v331_vista_installation_en.pdf

17.2.9. Import of the new opsi products

The downloaded packages has to be installed on the server to be available for the clients. The interactive installation of an opsi package happen with the aid of the order:

```
opsi-package-manager -i <package file name>
```

The following order install the downloaded packages successive:

```
opsi-package-manager -i *.opsi
```

The defaults of the javam package have changed (only 1.5 and 1.6 will be installed). In order to install this package with the changed defaults you should execute for this package:

```
opsi-package-manager -i javavm_1.6.0.12-2.opsi  --properties package
```

17.2.10. Activating the new support for the USB and HD-Audio driver

In order to activate the support for USB and HD-Audio drivers you should execute the **`create_driver_links.py`** script in every OS installation product directory (e.g /opt/pcbin/install/winxppro)

17.3. Update 3.2 to 3.3

17.3.1. Documentation

Please read the opsi 3.3 changes documentation in the opsi-manual.

17.3.2. Register of the opsi 3.3 repository

In order to avoid that an update to 3.3 happens accidentally the debian package for opsi 3.3 is in a specific repository. Delete in /etc/apt/sources.list the entry:
`deb http://download.uib.de/debian etch opsi3.2`
and put in:

For Debian sarge: (no update available. Upgrade to Etch)

For Debian Etch, Ubuntu Dapper/Edgy/Feisty (i386/amd64):

`deb http://download.uib.de/debian etch opsi3.3`

Execute **`apt-get update`**.

17.3.3. Put in the opsi debian packages

Put in the packages with following order:
`apt-get install opsi-depotserver opsi-configed`

`apt-get upgrade`

If you be asked on upgrading which version of a configuration file you wish to apply you should choose the newest version. If not you should know exactly what you do e.g. you don't choose the newest version because you want an other as the default File31-Backend.

The installation of the opsipxeconfd package may abort with a error which is caused by the old installation, not by the new package. In this case please execute:

`apt-get install opsi-depotserver`

If there are still errors, it should help to issue an

```
apt-get install -f
```

17.3.4. Checking the backend configuration

In the file /etc/opsi/backendManager.d/30_vars.conf is defined which backend manage of opsi be used (**BACKEND_FILE31, BACKEND_FILE, BACKEND_LDAP**).

The default backend is **BACKEND_FILE31.**

The backend **BACKEND_FILE** is deprecated - you should change to **BACKEND_FILE31.**

The backend **BACKEND_LDAP** is is not supported by opsi 3.3 yet - you should change to **BACKEND_FILE31** until LDAP-support is available.

In the entry **clientManagingBackend** may be controlled if opsi also assume the local DHCP configuration. This is sensible if the DHCP-server of the opsiserver will be used (default). The accordant entry is:

```
self.clientManagingBackend = [ BACKEND_DHCPD, BACKEND_FILE31 ]
```

If the local DHCP isn't used also the **BACKEND_DHCPD** not required:

```
self.clientManagingBackend = BACKEND_FILE31
```

For the hard- and software inventory you have since opsi 3.3 two possibilities: BACKEND_FILE31 or BACKEND_MYSQL. One of these you have to enter independent which backend is used normally:

```
self.swinventBackend = BACKEND_FILE31
self.hwinventBackend = BACKEND_FILE31
```

For the logging there is since opsi 3.3 a own Backend: BACKEND_FILE31. These you have to enter independent which backend is used normally:

```
self.loggingBackend = BACKEND_FILE31
```

After adapting the backend configuration the 'opsi-confd' has to be restarted:

```
/etc/init.d/opsiconfd restart
```

17.3.5. Import of the new opsi products

Fetch the actual necessary opsi packeges in the new package format:

```
cd /home/opsiproducts

wget -r -l1 -nc -nd -A '*.opsi' http://download.uib.de/opsi3.3/produkte/essential/upgrade
```

The downloaded package has to be installed on the server to be available for the clients. The interactive installation of an opsi package happen with the aid of the order:

```
opsi-package-manager -i <package file name>
```

The following order install the downloaded packages successive:

```
opsi-package-manager -i *.opsi
```

17.4. Update 3.1 to 3.2

17.4.1. Register of the opsi 3.2 repository

In order to avoid that a update to 3.2 happen accidentally the debian package for opsi 3.2 is in an own repository. Delete in /etc/apt/sources.list the entry:
`deb http://download.uib.de/debian etch opsi3.1`
and put in:

For Debian sarge: (no update available. Upgrade to Etch)

For Debian Etch, Ubuntu Dapper/Edgy/Feisty (i386/amd64):

`deb http://download.uib.de/debian etch opsi3.2`

Execute **`apt-get update`**.

17.4.2. Put in the opsi debian packages

Put in the packages with following order:

```
apt-get install opsi-depotserver; apt-get upgrade
```

If you be asked while the upgrade which version of a configuration file you will apply you should choose the newest version. If not you should know exactly what you do e.g. you don't choose the newest version because you want an other as the default File31-Backend.

17.4.3. Import of the new opsi products

Fetch the actual necessary opsi packages in the new package format:

```
cd /home/opsiproducts

wget -r -l1 -nd -A '*.opsi' http://download.uib.de/opsi3.2/produkte/essential/upgrade
```

The downloaded package has to be installed on the server to be available for the clients. The interactive installation of an opsi package happen with the aid of the order:

```
opsiinst <paketname>.opsi
```

The following order install the downloaded packages successive:

```
for paket in *.opsi; do opsiinst -f -q -k $paket; done
```

17.4.4. Checking the backend configuration

In the file /etc/opsi/backendManager.d/30_vars.conf is defined which backend manage of opsi be used (**BACKEND_FILE31, BACKEND_FILE, BACKEND_LDAP**). The default backend is **BACKEND_FILE31.** In the entry **clientManagingBackend** may be controlled if opsi also assume the local DHCP configuration. This is sensible if the DHCP-server of the opsiserver will be used (default). The accordant entry is:

```
self.clientManagingBackend = [ BACKEND_DHCPD, BACKEND_FILE31 ]
```

If the local DHCP isn't used also the **BACKEND_DHCPD** not required:

```
self.clientManagingBackend = BACKEND_FILE31
```

For the hard- and software inventory you have to enter the FILE31-backend independent which backend is used normally:

```
self.swinventBackend = BACKEND_FILE31
self.hwinventBackend = BACKEND_FILE31
```

After adapting the backend configuration the 'opsi-confd' has to be restarted.

17.5. Update 3.0 to 3.1

17.5.1. Register of the opsi3.1 repository

In order to avoid that a update to 3.1 happen accidentally the debian package for opsi 3.1 is in an own repository. Delete in /etc/apt/sources.list the entry:
`deb http://download.uib.de/debian sarge opsi3.0`
and put in:

For debian sarge (only i386):

`deb http://download.uib.de/debian sarge opsi3.1`

For debian Etch, Ubuntu Dapper/Edgy (i386/amd64):

`deb http://download.uib.de/debian etch opsi3.1`

Execute **`apt-get update`**.

17.5.2. Put in the opsi debian packages

Put in the packages with following order:
`apt-get install opsi-depotserver; apt-get upgrade`

If you be asked while the upgrade which version of a configuration file you will apply you should choose the newest version; if not you should know exactly what you do.

17.5.3. Adapt the configuration

Opsi 3.1 used per default the new backend “File31“. So you either adapt your configuration that your previous backend will used or the data base from th eold to the new backend convert. The classification of the opsi-backends to the different „functions“ will be defined in the file /etc/opsi/backendManager.d/30_vars.conf. If you want to use the file-backend further the corresponding section has to look like these:

```
self.defaultBackend          = BACKEND_FILE
self.clientManagingBackend = BACKEND_FILE
self.pxebootconfBackend     = BACKEND_OPSIPXECONFD
self.passwordBackend         = BACKEND_FILE
self.pckeyBackend            = BACKEND_FILE
self.hwinventBackend         = BACKEND_FILE
```

In these case it's important that the file-backend further on be loaded. In order to achieve this the line in the file /etc/opsi/backendManager.d/10_file.conf:

```
 'load': False
```

has to adapted in:

```
 'load': True
```

After changing the configuration the services opsiconfd and opsipxeconfd has to be started new. Execute the following order:

```
/etc/init.d/opsiconfd restart; /etc/init.d/opsipxeconfd restart
```

Should you decide to use the File31-backend the files has to be converted. **Before you convert your system make a backup of your system!** For the conversion of files the program opsi-convert will used. The order for a conversion from File- to File31-backend is:

```
opsi-convert File File31
```

After a conversion between the two file based backends the file /etc/opsi/pckeys should be corrected manually because both backends are using this file but the File31-backend requires entries with fully qualified domain names, e.g.:

```
clientname.domain.tld:1bad67e3c6955ccac891f58ca31ed37e
```

In contrast, the classic File-backend has lines with simple host names, e.g.:

```
clientname:1bad67e3c6955ccac891f58ca31ed37e
```

17.6. Update 2.5 to 3.0

17.6.1. Register of the opsi 3-repository

In order to avoid that an update to 3.0 happens accidentally the debian package for opsi 3.0 is in a special repository. Delete in /etc/apt/sources list the entry

deb http://download.uib.de/debian sarge main

and put in:

deb http://download.uib.de/debian sarge opsi3.0

Execute **apt-get update**.

17.6.2. Put in the opsi Debian package

Put in the package with the order

apt-get install opsi-depotserver opsi-configed opsi-linux-bootimage

These order should create the following output

```
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  opsi-reinstmgr opsi-utils opsiconfd python python-crypto python-json
  python-ldap python-newt python-opsi python-pam python-pyopenssl
  python-twisted python2.3 python2.3-crypto python2.3-ldap python2.3-pam
  python2.3-pyopenssl python2.3-twisted python2.3-twisted-bin sun-j2re1.6
Suggested packages:
  python-doc python-tk python-profiler slapd python-gtk2 python-glade-1.2
  python-glade2 python-qt3 libwxgtk2.4-python python2.3-doc python2.3-profiler
  python-ldap-doc pyopenssl-doc
Recommended packages:
  python-serial python2.3-iconvcodec python2.3-cjkcodecs
  python2.3-japanese-codecs
The following NEW packages will be installed:
  opsi-configed opsi-reinstmgr opsi-utils opsiconfd python python-crypto
  python-json python-ldap python-newt python-opsi python-pam python-pyopenssl
  python-twisted python2.3 python2.3-crypto python2.3-ldap python2.3-pam
  python2.3-pyopenssl python2.3-twisted python2.3-twisted-bin sun-j2re1.6
The following packages will be upgraded:
  opsi-depotserver opsi-linux-bootimage
2 upgraded, 21 newly installed, 0 to remove and 0 not upgraded.
Need to get 88.0MB of archives.
After unpacking 120MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

(.....)

The package opsiconfd need some entries to create a SSL-certificat:

```
Setting up opsiconfd (0.9-1) ...
Generating a 1024 bit RSA private key
.....................................................................++++++
..........++++++
writing new private key to '/etc/opsi/opsiconfd.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:de
State or Province Name (full name) [Some-State]:Rheinland-Pfalz
Locality Name (eg, city) []:Mainz
Organization Name (eg, company) [Internet Widgits Pty Ltd]:uib
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []: opsidepot.uib.local
Email Address []:info@uib.de

The user `pcpatch' is already a member of `shadow'.
Starting opsi config service...   (done).
```

(....)

After you finished these chapter go on with 'Inspect the configurations' to 'Put in the minimal opsi-products'.

17.7. Update 2.4 to 2.5

The Update is easy

```
# Informationen über neue Pakete holen
apt-get update
# altes depotserver paket removen
apt-get remove opsi-depotserver
# neue pakete installieren
apt-get install opsi-depotserver
apt-get install opsi-webconfigedit
apt-get install opsi-inied
# Notwendige opsi-Pakete holen
wget -r -l 1 -nd -nH --cut-dirs=5 -np -N -R "*.html*" \
www.uib.de/www/download/download/opsi-pakete/essential
# notwendige opsi-Pakete installieren
opsiinst win2k.cpio.gz
opsiinst winxppro.cpio.gz
opsiinst opsi-winst.cpio.gz
opsiinst preloginloader.cpio.gz
opsiinst softinventory.cpio.gz
opsiinst opsi-adminutils.cpio.gz
opsiinst javavm.cpio.gz
```

17.8. Update 2.x to 2.4

The update is time-consuming because the versions before 2.4 working not with debian-packages (or only in parts) and some things have to be installed new.

Specially it's a system software update from 'Debian Woody (3.0)' to 'Debian Sarge (3.1)' and from 'Kernel 2.4' to 'Kernel 2.6'. If you don't know an update with 'apt-get dist-upgrade' and haven't possibilities for testing, it will be better to reinstall the server or to ask some experts (for example uib).

After warning you now the important facts:

Adapt the file '/etc/apt/sources.list' to install the debian-package out of 'stable' and get extra sources.

Here an example

```
#Standard debian Quellen:
deb http://sunsite.informatik.rwth-aachen.de/ftp/pub/Linux/debian/ stable main
non-free contrib
deb-src http://sunsite.informatik.rwth-aachen.de/ftp/pub/Linux/debian/ stable
main non-free contrib
deb http://non-us.debian.org/debian-non-US stable/non-US main contrib non-free
deb http://security.debian.org/ stable/updates main
#Hier gibts den FreeNX-Server:
deb http://www.linux.lk/~anuradha/nx/ ./
#Alternative Samba Quelle:
deb http://ftp.sernet.de/pub/samba/ debian/
#opsi-Pakete:
deb http://www.uib.de/www/download/download/debian sarge main
```

Update with 'apt-get' the databank package. If this isn't possible, you may have to put a proxy in the file '/etc/apt/apt.conf' or delete one.

Before you can start 'dist-upgrade', you have to correct some dependencies:

```
apt-get install libcrypt-blowfish-pp-perl
apt-get install apache-common
```

Update now the system software:

```
apt-get dist-upgrade
```

Edit the '/etc/login.defs' and put '/opt/bin' in the path.

To proceed on:

```
apt-get install kernel-image-2.6.8-2-686
apt-get install kernel-source-2.6.8
apt-get remove opsi-depotserver
#optional (bei Neuinstallation vorhanden)
apt-get install xfce4
apt-get install wget
apt-get install traceroute
apt-get install nxserver
#-> configuration: custom keys
apt-get install mozilla-firefox
```

Now you have done the most work and could go to chapter 'Installation of a debian (Sarge) system with apt-get'.

18. History

18.1. Difference between opsi version 3.3.1 and version 3.3

18.1.1. What's new at opsi 3.3.1

- Support for Debian Lenny
- Redesigned packages for OS installation
  containing:
    - unified structure
    - Support for HD-Audio and USB driver at the simplified driver integration
    - Support for multiple i386 directories
- For customers of the opsi vista support:
    - Enhanced support for 64 bit versions of Vista/2008
    - New preloginloader 3.4 not only for Vista
    - Vista service packs and hotfixes products
- The required action 'alwas' is now also supported at netboot products
- New Virtual Machine based on Debian Lenny
- topical manuals
- diverse bug fixes
- The support for the opsi 2.x/3.0 File-Backend has ended
    - User of the File-Backend should **convert their configuration data to the File31-Backend before updating to opsi 3.3.1**

18.1.2. What you should read in case of a upgrade to opsi 3.3.1

At this manual.

- 3.4 Tool: opsi-package-manager: (de-)installs opsi-packages on page 28
- 6.1 opsi standard products on page 59
- 7.1.8 Structure of the unattended installation products on page 70
- 7.1.9 Simplified driver integration with symlinks on page 70
- 13 Adapting the opsi preloginLoader to your Corporate Identity (CI) on page 130

At the opsi vista installation manual:

- preloginloader 3.4

18.2. Difference between opsi version 3.3 and version 3.2

18.2.1. Overview

- support of multiple depot-servers
    - The multi-depotshare extension gives you the opportunity of a central administration for different locations
    - At every location an opsi depot-server provides the shares and the TFTP/PXE bootserver functionality, while the administration and the configuration data backend is still handled by the central opsi-server.
    - A decentral opsi depot-server can be easily installed and integrate with the central configuration server.
    - Up to now, these new functions are only implemented for the 'File31-Backend'. The new LDAP-Backend is coming soon.
    - The multi-depotshare extension is part of the 'Professional Edition'. It is open source, but for getting support you will need a professional support contract.

- opsi-package-manager: The new package administration tool
    - Install and deinstall opsi packages on one ore more opsi depot-servers. (Replacing the deprecated commands opsiinst and opsiuninst)
    - Lists which packages installed on which server
    - Shows installation differences between different depot-servers
    - Extracts opsi packages for modification purpose.
- opsi-configed enhancement
    - support of multiple depots
    - client MAC-address now editable
    - enhanced client creation dialog
- MySQL-backend for inventory data with history function
- opsi-winst extensions
    - opsi-winst now has a new skin, and has become skinnable: other skins may be easily constructed without programming, e.g. for integration in a CI concept.
    - New Command 'ExitWindows /ShutdownWanted' for shutdown after all installations have finished
    - New copy option suppressing automatic reboots if the target file was in use
    - New function for retrieving the system locales
    - New function for retrieving the version informations of windows files
    - New function for catching the exit code of called programs
    - New secondary section for the execution of scripts with arbitrary external interpreters

    - New function for retrieving strings from a map (string list with elements key=value)
    - Easier building of partials lists
    - Run time debugging support for script execution
- Enhanced opsi-agent 'preloginloader'
    - no reboot on preloginloader rollout with the opsi-preloginloader-deploy script.
    - Faster reaction if the opsi-server is missing
    - inital installation of the login blocker as default
    - no more permanent local user pcpatch
    - Automatic and enhanced MAC-address detection
    - enhanced installation program
- extended automatic driver integration used for the OS-Installation
- extended opsi-admintools functionality
    - Up-to-date opsi-configed as application
    - Extracting opsi-packages under Windows:: 7-zip
    - Editor (not only) with opsi-winst syntax-highlightning: jedit
    - SSH-Terminal: Putty
    - Compare and merger text files:: WinMerge.
    - XML-Editor: Pollo
    - XML-Diff-Tool
    - LDAP-Explorer: JXplorer

    - Tool for interactive setups with recorded answers: Autohotkey.
    - Setup Switch detector: Ussf
- new linux bootimage
    - Up-to-date kernel
    - NTFS write support
- Enhanced WakeOnLAN function
- some bugfixes
- opsi 2.x/3.0 File-Backends are now deprecated
    - users of this Backend should change to the File31-Backend
- New documentations and installation media

Vocabulary:

config-server	the functionality which provides management and storage of opsi configuration data
depot-server	the functionality which provides software depot shares and tftpservice for opsi based PXE boot.
opsi-server	a server which provides opsi functionality most times -> config-server and -> depot-server.

18.2.2. What you should read in case of a upgrade to opsi 3.3

At this manual.

- 3.2.3 Depot selection on page 17
- 3.2.5 Client processing / WakeOnLan / Create a Client / Move a Client on page 19

- 3.4 Tool: opsi-package-manager: (de-)installs opsi-packages on page 28
- 7.1.9 Simplified driver integration with symlinks on page 70
- 10 opsi-server with multiple depots on page 111
- 12.3 MySQL-backend for inventory data on page 121

In the opsi-winst manual:

- ExitWindows /ShutdownWanted

18.2.3. Migration to opsi V3.3.1

The migration of the opsi environment from opsi V3.3 to opsi 3.3.1 is described in the opsi depot server installation handbook.

18.3. Difference between opsi version 3.2 and version 3.1

18.3.1. Overview

- Upgraded hardware inventory
    - The opsi-product 'hwaudit' detects hardware information per WMI and reports it to the opsi depot server
    - opsi-configed displays the current hardware inventory sorted by device classes
    - Selection of clients by certain hardware criteria like e.g. the size of main storage
    - Provides server-sided extensions to transfer hardware inventory data via the web service and save it to the backend data base
    - Ability of the opsi-wInst to execute python scripts out of a wInst-section
- Upgraded software inventory

    - The opsi-product 'swaudit' collects software information from the client registry and reports it to the opsi depot server
    - opsi-configed displays the current software inventory
    - Provides serversided extensions to transfer the software inventory data via the web service and save it to the backend data base
- Accelerated performance of the unattended installation of WinXP/2k (without using DOS anymore)
- Upgraded process to save and restore NTFS-images
- Other netboot products:
    - wipedisk: Fast or very secure data deletion from the hard disk
    - memtest: Test the client memory
- Several bugfixes
- Updated documentation and installation media:
    - opsi V3.2 installation handbook
    - opsi V3.2 handbook
    - opsi-wInst handbook
    - opsi-wInst quick reference
    - Virtual opsi V3.2 opsi depot server for Vmware
    - Installation CD for opsi V3.2 opsi depot server

18.3.2. What you should read

Opsi V3.2 brings along some news you should be familiar with. Please first read this chapter and then refer for further information:

- Chapter: 'swaudit' and 'hwaudit': Products for the hard- and software inventory
- Chapter about netboot products like ntfs-image, wipedisk and memtest
- The opsi integration handbook: opsi V3.2 has been added to this handbook
- Updated wInst-handbook

18.3.3. Migration to opsi V3.2

The migration of the opsi environment from opsi V3.x to opsi 3V.2 is described in the opsi depot server installation handbook.

18.4. Difference between opsi Version 3.1 and Version 3.0

18.4.1. Overview

- Integration of boot image based products into the standard data management
    - Boot image products like OS-installation, hardware inventory, create images or restore images are now integrated in the normal data configuration like other products and can be administrated the same way
    - The opsi-reinstmgr is replaced by the opsipxeconfd, which gets a direct access to the opsi configuration by the opsi-Python-Library.
- Extensions of the opsi-configed
    - Information about installed software and package versions of a product are displayed and evaluated
    - Basic opsi configurations (Generalconfig) are now editable
- New scripts for the initial rollout of the opsi-preloginloader
    - opsi-deploy-preloginloader
      Starts the installation of the opsi-preloginloader from the server side. The admin password of the client and an open C$- and admin file share are required

    - setup_service.cmd
      If the requirements for the script opsi-deploy-preloginloader aren't met, the administrator can start a script on the client side to generate the client entry per opsi-service and install the preloginloader on the client side (by providing username/password of an opsi-admin in the script)

- Simplified driver integration based on the PCI Vendor- and Device-IDs
    - A new script passes just the required additional drivers to the Windows installation process, so it doesn't need to scan all the available drivers anymore

- Improvements of the opsi-preloginloader / opsi-wInst
    - The opsi-preloginloader now is more stable in situations with a broken name resolution
    - Bugfix regarding the support of English systems
    - opsi-wInst function for identification of the system language at run time for supporting multilingual packages.
    - opsi-wInst support request of the opsi-Service from within wInst-scripts
- Data conversion between different backends, e.g. from file-backend to the LDAP-backend
- Some more adaptations on the linux standard base with the new file3.1-backend

18.4.2. What you should read

Opsi V3.1 brings along some news you should be familiar with. As an introduction please first read this chapter.

And further on refer to:

- Chapter 4.1.3 Subsequent installation of the opsi-preloginloader
- Description of the backend you are using

- Description of the opsi-configed
- Chapter Driver integration in a Windows installation
- wInst-hand book

18.4.3. Backend

opsi V3.1 supports the following backends:

- File
  Data based backend. This is backward compatible with opsi V2.x and being located in /opt/pcbin/pcptch it is not conform to the 'Linux Standard Base'.
- File3.1
  Data based backend. This is incompatible to opsi2.x and being located in /var/lib/opsi it is conform to the 'Linux Standard Base'.
  The essentially differences to the 'file'-backend are:
  - Aggregation of the product administration of 'normal' products and bootimage based (localboot- and netboot products) in one data file per client
  - Separation of current status and requested action for each software product
- LDAP
  Standard Open-LDAP opsi-Backend
- Univention-LDAP
  Backend of the opsi special edition opsi4ucs

The default backend for a new opsi installation is: File3.1

18.4.4. Migration to opsi V3.1

The migration of the opsi environment from opsi V3.0 to opsi V3.1 is described in the opsi depot server installation handbook.

18.5. Differences of opsi version 3 to version 2

18.5.1. Overview (What you should read)

The essentials about new features and technologies of opsi V3 are described in the following chapters:

This one: 18.5 Differences of opsi version 3 to version 2

Chapter Fehler: Referenz nicht gefunden Fehler: Referenz nicht gefunden

Chapter 3.2 Tool: opsi V3 opsi-Configed

Chapter 3.3Fehler: Referenz nicht gefunden Tool: opsi V3 opsi-Webconfiged

Chapter 3.5 Tool: opsi V3 opsi-admin

For the new distribution packet format in opsi V3 read the chapter 10.2 in the opsi integration hand book.

18.5.2. Conceptual

In opsi V2 the complete data storage has been file based. All opsi components operated directly on these text files.

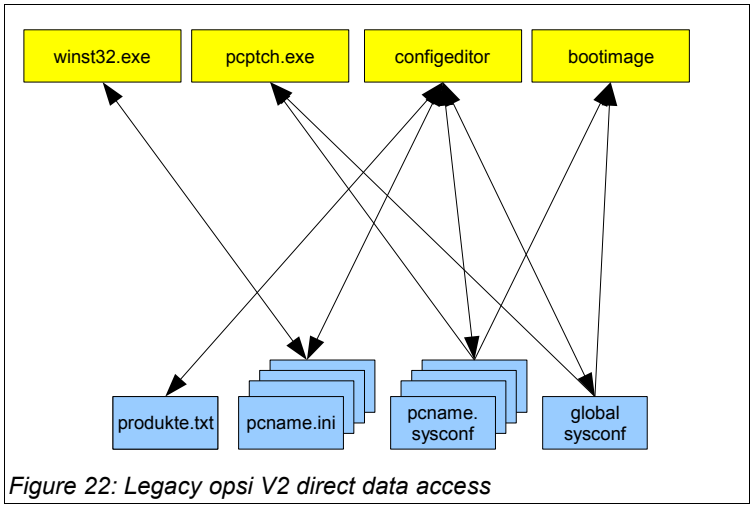


Figure 22: Legacy opsi V2 direct data access

In opsi V3 the opsi components don't operate directly on the data storage but insteaad use a web service which is provided by a opsi configuration daemon. Only this daemon reads and writes to the data storage.

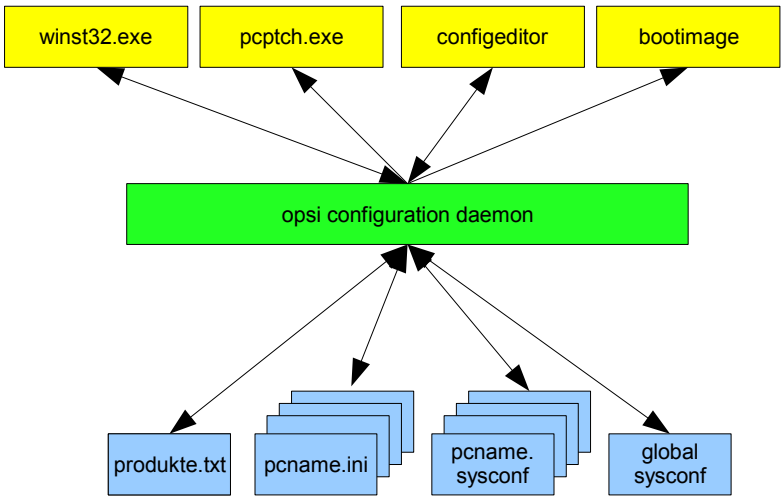


Figure 23: opsi V3 using the web service for data access

Using this daemon makes it quite easy to use different types of data storage. So opsi V3 comes with an optional LDAP based data storage.

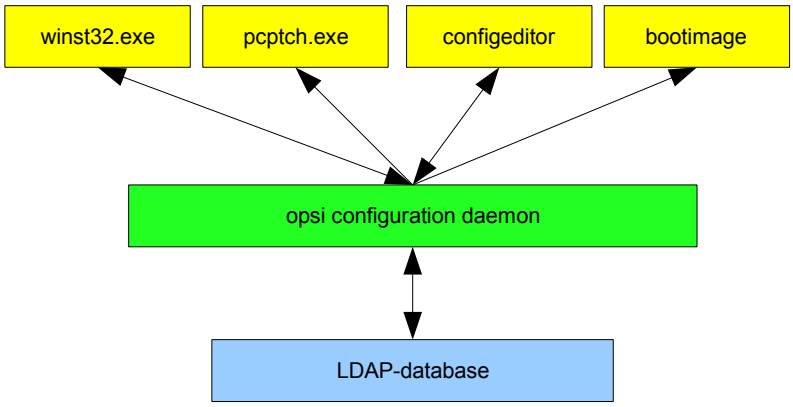


Figure 24: use of different storage systems (LDAP) by the daemon

For backward compatibility reasons opsi winst.exe and pcptch.exe also have a 'classic' mode with direct access to a text file based data storage.

The realization of this concept is done in opsi V3 via a new opsi-Python-Library. This python based library provides an opsi configuration API which is independent from the actual type of data storage. This API is also provided by a web service which uses the JSON standard. This web service is implemented as part of the program opsi-configed. The program opsi-admin provides a command line interface to the API for shell scripting.

Another part of the opsi-Python-Library implements the concrete access to the different types of data storage (backends) which is configured by the backend manager.

18.5.3. Improvement of the handling

Beside the technical changes there are a lot of improvements of the handling of opsi V3.

- Configuration Editor: opsi-configed
    - Group management:
        - Multiple selection and configuration of clients
        - Save and load different groups of clients for configuration
        - Using filters for selecting groups of clients on the fly (by criteria like installed software)
    - WakeOnLAN for selected client groups from within the configuration editor
    - Client list may be sorted by name, description and 'last seen by opsi'
    - Changed the opsi V2 installation switches into separate status informations for the actual installation status and the requested action
    - Product list may be sorted by installation status and requested action
    - The opsi-configed is also provided as web applet

    - Improved view on the hardware information
    - Simple creation or deletion of clients
- New packet format for installation of opsi products on the opsi depot servers
    - Easy menu driven creation of new packets
    - Informations about software version, packet version and custom additions are stored in the packet. They are also shown by the packet name and in the destination directory on the depot server. This will help you in your product life cycle management.
    - Creation and installation of opsi packets is now possible without root rights
    - The commands for handling legacy packets are still available as opsiinstv2 and makeproductfilev2
- command line tool 'opsi-admin' for script driven opsi configuration
- opsi4ucs: opsi for the Univention Corporate Server (UCS)
    - Integration of opsi data storage to the UCS-LDAP
    - Integration of opsi configuration to the 'Univention Admin Interface'
    - There is a special manual: 'opsi4ucs'

18.5.4. Vocabulary

For opsi V3 some new definitions are used and some have changed since opsi V2.

Here are some of the important definitions:

action request	Action which will be executed next. something like: 'setup', 'deinstall' or 'update'. -> installation status
backend	opsi V3 may use different types of data storage (backends) like File or LDAP. Which and how these backends are used is configured by the -> backend manager.

backend manager	Program / configuration file to configure the different backends
clientId	unique identifier for a client using the 'full qualified hostname' e.g. dpvm02.uib.local
hostId	unique identifier for a computer using the 'full qualified hostname' e.g. dpvm02.uib.local
installation status	actual installation status of a product on a specific client, typically something like: 'installed' or 'not installed'. -> action request
LastSeen	Time stamp of the last client call to the opsi web service
localboot product	An opsi packet which will be installed by the opsi-preloginloader
netboot product	An opsi packet which will be handled by starting a boot image
opsi-admin	opsi V3 command line interface for opsi-Configuration
opsiHostKey	see pckey
opsi-Configed	opsi V3 graphical configuration tool as Java Application and Applet
opsiconfd	Daemon that provides the opsi configuration API as JSON based web service
product properties	additional client specific product configuration
productId	unique identifier for an opsi product. Special characters (beside '-') are not allowed. Example: acroread
product name	Full name of the software product. Example: 'Adobe Acrobat Reader'
server product	An opsi product which installs something on the server which is no client software

18.5.5. Migration to opsi V3

The migration of the opsi environment from opsi V2 to opsi V3 is described in the opsi depot server installation handbook.

19. Glossary

action request	Since opsi V3 the installation status and the next scheduled action (action request) are handled as separate. Typical action requests are 'setup', 'deinstall' and 'update'. -> installation status.
backend	opsi V3 may use different types of data storage (backends) like 'File' or 'LDAP'. These backends are managed by the -> backend manager.
backend manager	Program / configuration file for handling the actual data storage (backend).
bootp	Bootstrap protocol, first used to boot terminals in Unix environments. Allows a client to request configuration data (i.e. network address), often via -> Bootprom from a server.
bootprom	Read only memory installed on a -> NIC or main board (PROM: Programmable Read Only Memory). The BIOS of a PC can execute the code stored in the bootprom at boot time. The purpose is to load configuration data from a server on the network. The protocol for this usually is ->bootp or ->DHCP.
clientId	Unique client name as the 'full qualified hostname', which is the client's IP-name including the domain (e.g. 'dpvm02.uib.local').
DHCP	Dynamic Host Configuration Protocol: an extension of the ->bootp-protocol allowing dynamic IP address assignment.
ftpd	File Transfer Protocol-Daemon: Daemon for both ends of the ftp-protocol. Allows remote logins and file transfer from and to remote hosts. **telnetd** (Telnet-Daemon) provides (insecure) terminal connections from remote machines.

GINA	Graphical Identification and Authentication is a Microsoft Windows program handling the user login. The default is 'msgina.dll'. For modifying the login process, additional GINAs can extend the msgina. The opsi-loginblocker (for preventing the user login during software installation) is a GINA extension based on the source of 'pgina.dll' (http://pgina.xpasystems.com).
GNU	The recursive abbreviation GNU stands for "GNU's not Unix". The GNU-project was initiated in 1983 by Richard Stallman, founder of the Free Software Foundation, to develop a free Unix like OS. This project is still in progress and originated a lot of GNU-tools, that allowed the development of LINUX. Therefore Linux is often referred to as GNU/Linux.
GUI	Graphical User Interface.
hostId	Unique ID of a computer by using the 'full qualified hostname', which is the 'IP-name' including the domain (e.g. 'dpvm02.uib.local')
inetd	Internet Daemon: is the master service of some other daemons, which are for instance bootpd, ftpd, tftpd and telnetd. These daemons are started on request by the inetd according to the inetd configuration file '/etc/inetd.conf'.
Installation status	Since opsi V3 the installation status and the next scheduled action (action request) are handled as separate. The installation status is usually 'installed' or 'not installed'. -> action request
IP address	IP (Internet Protocol) address is an unique address within the internet or subnet. The IP address is a 32-bit number composed of the network address and second the machine-address within the network. Usually the 32-bit number is written as four decimal numbers (0..255) separated by a dot (e.g. 194.31.284.12).

	Dependent on the network size the network is classified as class A, B or C. In class A networks (for very large networks) the first number (1..126) addresses the network itself and the three remaining segments represent the machine's address. Class B networks use the first two numbers to address the network (the first number must be 128..191) and the last two numbers to specify the machine. In a class C network three numbers address the network and just the last segment is used to specify the machine. All three classes have an address range (i.e. 192.168. for private networks) which is not routed to the internet. This class structure is somehow outdated since classless inter domain routing became practice to make better use of the limited resources of IPv4 addresses.
JSON	**JSON** short for **J**ava**S**cript **O**bject **N**otation is a compact data exchange format. The data are easy to read for people and for machines. Source: http://de.wikipedia.org/wiki/JSON and www.json.org
LastSeen	Time stamp of the last client connect to the opsi service.
localboot product	An opsi packet which is installable by the opsi-preloginloader.
MAC address	The 'Media Access Control address' is an unique identifier attached to the network adapter and is transferred with every data packet. With this address the computer (respectively its network card) can be identified worldwide and can be mapped to an →IP-number. The MAC address is composed from 6 hexadecimal numbers (0..FF) separated by colon (e.g. 00-08-74-4C-7F-1D). The first 3 numbers identify the manufacturer of the network adapter.
netboot product	An opsi packet which is installable by a bootimage.

NIC	Network Interface Controller – hardware to connect the network
opsi-admin	opsi V3 command line interface for opsi configuration
opsi-Configed	opsi V3 configuration tool (Java application and applet)
opsi-preLoginLoader	- opsi service on Windows clients to install software packets.
opsiconfd	opsi configuration daemon - provides the opsi configuration API as a JSON based web service.
opsiHostKey	see pckey
pckey	A string assigned to the client during the (preloginloader-) installation, which is also saved on the server. The pckey is used for client authentication and not accessible for standard users. →opsiHostKey
PDC	Primary Domain Controller: primary authentication server of a Microsoft network.
pgina	-> GINA
preloginloader	-> opsi-preLoginLoader
product properties	A product can be configured at installation time by evaluating the product properties, which are client specific settings. These could for instance indicate, whether some additional modules should be installed on that client. Or the property could specify an attribute to patch the installation in some way or another.
product ID	Unique name of an opsi-product (A..Z, numbers and hyphen, no spaces allowed). In opsi V2 this is often used as a synonym for -> product name, which has a different meaning with opsi V3. Example for a product Id: acroread

product name	In opsi V3 this is the full name of a product (allowing blanks). Example for a product name: 'Adobe Acrobat Reader'.
PXE	Preboot eXecution Environment: common standard for bootproms. Usually ->DHCP (not ->bootp) is used with PXE bootproms.
SAMBA	Open source software to provide services on Unix/Linux servers for the ->SMB protocol (used by Microsoft clients).
Server product	An opsi product which is executing installations on the server only (containing no installable client software).
SMB	Server Message Block: Protocol by Microsoft to support network shares and authentication. Recently also named CIFS (Common Internet File System).
Subnets	In case of large local networks it often makes sense to divide it into subnets. To do this, an arbitrary sized part of the machine address within the IP is defined to be the subnet. In this case the IP address has three parts: network, subnet and machine. The subnet mask determines which part of the IP remains machine specific by setting these bits in the subnet mask to zero and the bits for the network and subnetwork to one.
TCP/IP	Transmission Control Protocol / Internet Protocol: is originated from the Unix world and has become the base protocol for all kinds of internet communication. All the services for web, email, file transfer etc. are based on TCP/IP.
tftpd	tftp (Trivial File Transfer Protocol) is a file transfer protocol (based on -> TCP/IP) without interactive login. The file transfer is managed by the tftpd (tftp daemon). For security reasons tftpd has limited access to the file system and may only transfer files from a dedicated directory (usually '/tftboot'). The files to be transferred

must be fully accessible for all users. The opsi clients are using tftp to fetch boot menus and bootimages from the server.

20. Table of Figures

Table of Figures

21. Additions and Changes

Additions and changes in this hand book.

21.1. opsi 2.4 to opsi 2.5

- Usage of https in the web configuration editor
- Chapter on driver integration for the automatic software OS installation
- Chapter for subsequent installation of the preloginloader
- References to opsi-wiki
- References to the opsi bootimage handbook
- List of the opsi log files

21.2. Additions opsi 2.5 (9/25/06)

- Option 'askBeforeInst' in 'global.sysconf' has been moved from the [general] section to the product section
- Description of the switch 'textcolor' (wInst customizing)

21.3. Additions opsi 2.5 / opsi 3.0 (12/8/06)

- Registry entry 'button_stopnetworking' is in 'opsi.org/pcptch'

21.4. Additions opsi 3.0 (1.2.07)

- v3 Chapter: Differences between opsi Version 3 to Version 2
- v3 Chapter: Programs in /opt/bin
- v3 Extension: Configuration files in /etc/opsi
- v3 New entries in the registry
- v3 Extensions: Configuration files for the software distribution: <pcname>.ini

- v3 Chapter: *.sysconf-files
- v3 Chapter: files in /etc/init.d
- v3 Chapter: /etc/group
- v3 Chapter: Tool: opsi-admin
- v3 Chapter: Tool: opsi V3 opsi-Configed
- v3 Chapter: Tool: opsi V3 opsi-Webconfigedit
- v3 Chapter: Tool: opsi V3 opsi-admin
- v3 Chapter: Log files in /var/log and /var/log/opsi
- v3 Additions to the glossary
- v3 Extension: Subsequent installation of the opsi-PreLoginLoader: Every client needs an entry in /etc/opsi/pckeys

21.5. Additions opsi 3.0

- v3 12.4.07: LDAP chapter

21.6. Additions opsi 3.1 (15.6.07)

- v3.1 Chapter differences 3.1
- v3.1 Chapter File31 Backend
- v3.1 Deleted: Tool reinstmanager
- v3.1 opsi-admin task setPcpatchPassword
- v3.1 opsi-admin client bootimage activate
- v3.1 Actualized: description of the opsi-configed
- v3.1 Actualized: chapter on the preloginloader rollout

v3.1 Actualized: chapter on driver integration

v3.1 opsi-admin: new methods:
method authenticated
method checkForErrors
method deleteProductProperties productId *objectId
method deleteProductProperty productId property *objectId
method deleteServer serverId
method getHost_hash hostId
method getNetBootProductIds_list
method getPossibleProductActionRequests_list
method setPXEBootConfiguration hostId *args
method setPcpatchPassword hostId password
method unsetPXEBootConfiguration hostId

21.7. Additions opsi 3.2 (21.11.07)

Actualized: the chapter 'Simplified driver integration with symlinks' for driver integration (**`download_driver_pack.py`** and **`preferred`**)