Documentation

opsi
Version 3.2

Automated software distribution

opsi depot server

Automated operating system installation

Hard- and software inventory

open pc server integration:
Boot- and Installation server for workstation PCs

Status: 1/22/09

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 (in its complete version) on UNIX-/Linux-servers. They are used for remote installation and maintenance of the client OS and the client software packets as well as for managing user accounts and user directories ("PC-Server-Integration"). It is composed of several configurable modules, which are based as far as possible on free available tools (*GNU*-tools, *SAMBA* etc.) and can be used stand alone or all together. The complete system all together is named **opsi** (Open PC-Server-Integration) and with its modularity and configurability is a very interesting solution for the administration challenges of a large computer park.

2.1. Experience

opsi, now in its version 3.2, 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
- comfortable control via the opsi management interface

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

2.3. Differences of opsi version 3 to version 2

2.3.1. Overview (What you should read)

The essentials about new features and technologies of opsi V3 are described in the following chapters:

This one: 2.3 Differences of opsi version 3 to version 2

Chapter

Chapter 3.2 Tool: opsi V3 opsi-Configed

Chapter 3.3 Tool: opsi V3 opsi-Webconfiged

Chapter 3.4 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.

2.3.2. Conceptual

In opsi V2 the complete data storage has been file based. All opsi components operated directly on these text files.

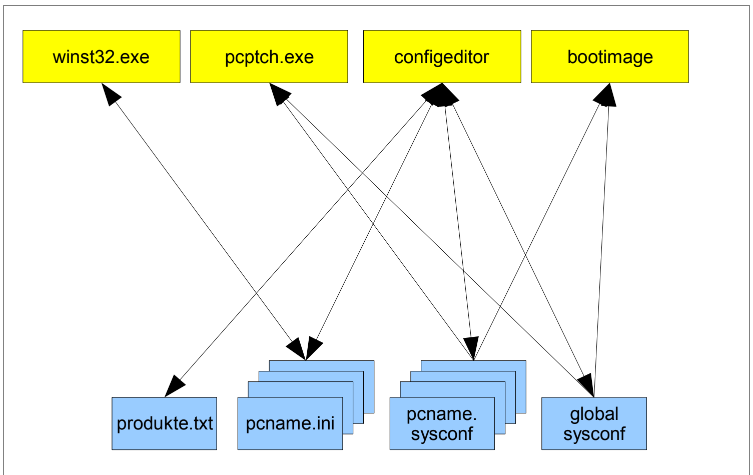


Figure 1: Legacy opsi V2 direct data access

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.

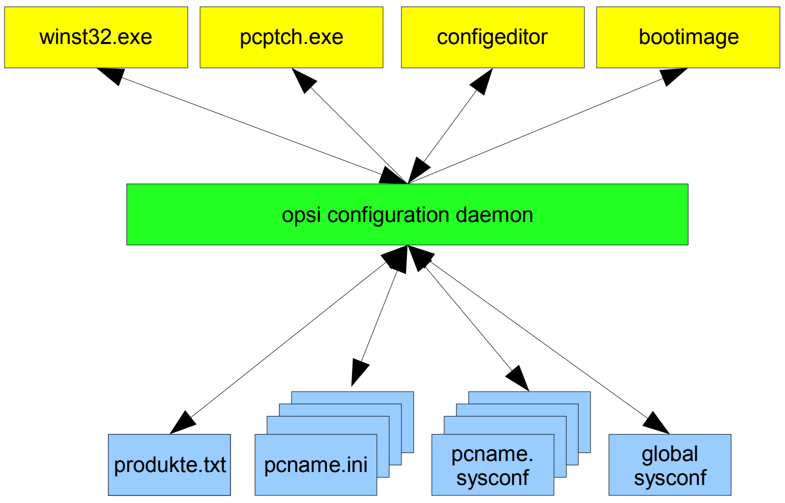


Figure 2: opsi V3 using the web service for data access

Using this daemon makes it quite easy to use different types of data storages. So opsi V3 comes with an optional LDAP based data storage.

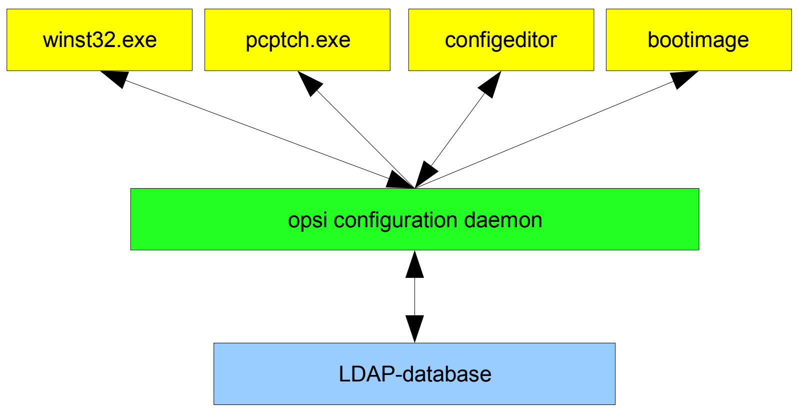


Figure 3: use of different storage systems (LDAP) by the daemon

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.

2.3.3. 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
    - Simplecreation 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'

2.3.4. 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

2.3.5. 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.

2.4. Difference between opsi Version 3.1 and Version 3.0

2.4.1. 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

2.4.2. 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

2.4.3. 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

2.4.4. 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.

2.5. Difference between opsi version 3.2 and version 3.1

2.5.1. 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

2.5.2. 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

2.5.3. 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.

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`**.

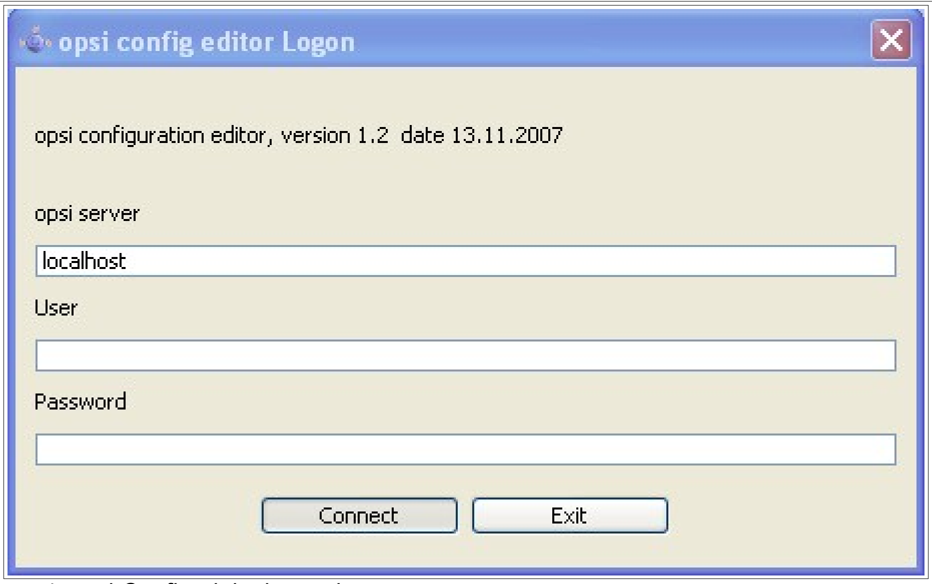
The help option **`java -jar configed.jar --help`** shows the available command line options.

```
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



opsi configuration editor, version 1.2 date 13.11.2007

opsi server

localhost

User

Password

Figure 4: opsi-Configed: login mask

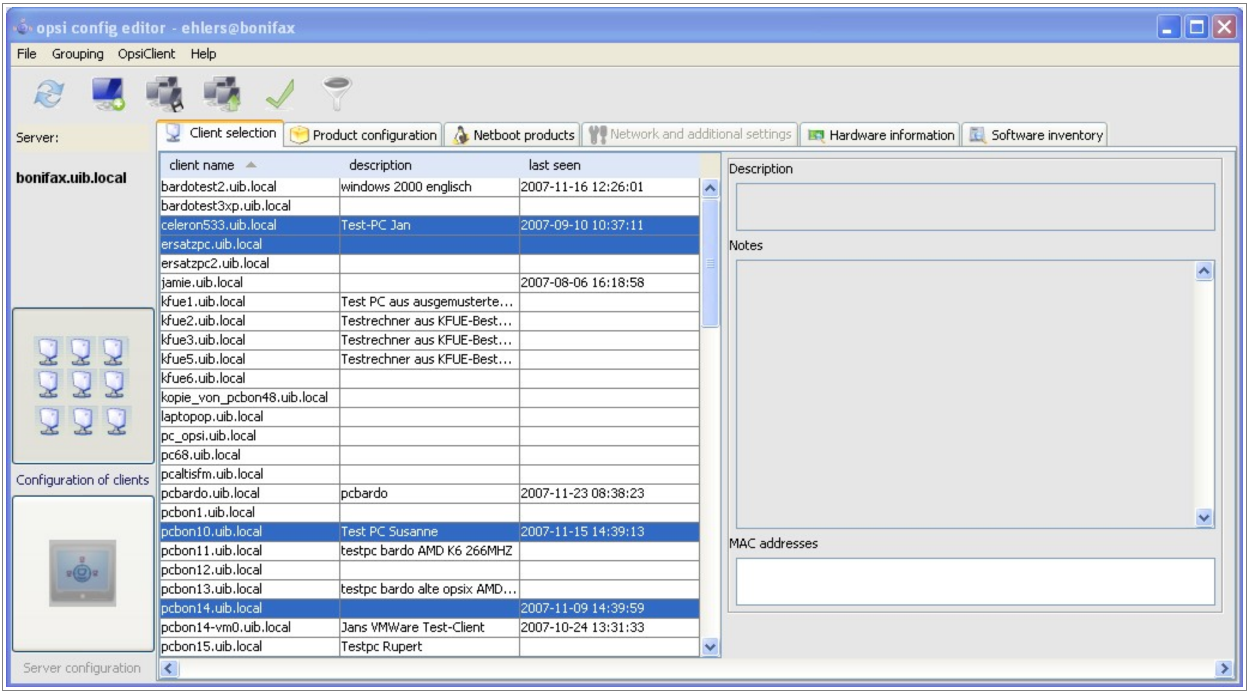
At login time the opsi-configed tries to connect the opsi server via https. The login is done with the given parameters opsi server[:Port] (default port 4447 – opsiconfd) and the User/Password of the opsi depot server account. For a successful login the provided user has to be a member of the unix-group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

3.2.3. 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



client name	description	last seen
bardotest2.uib.local	windows 2000 englisch	2007-11-16 12:26:01
bardotest3xp.uib.local		
celeron533.uib.local	Test-PC Jan	2007-09-10 10:37:11
ersatzpc.uib.local		
ersatzpc2.uib.local		
jamie.uib.local		2007-08-06 16:18:58
kfue1.uib.local	Test PC aus ausgemusterte...	
kfue2.uib.local	Testrechner aus KFUE-Best...	
kfue3.uib.local	Testrechner aus KFUE-Best...	
kfue5.uib.local	Testrechner aus KFUE-Best...	
kfue6.uib.local		
kopie_von_pcbon48.uib.local		
laptopop.uib.local		
pc_opsi.uib.local		
pc68.uib.local		
pcaltisfm.uib.local		
pcbardo.uib.local	pcbardo	2007-11-23 08:38:23
pcbon1.uib.local		
pcbon10.uib.local	Test PC Susanne	2007-11-15 14:39:13
pcbon11.uib.local	testpc bardo AMD K6 266MHZ	
pcbon12.uib.local		
pcbon13.uib.local	testpc bardo alte opsix AMD...	
pcbon14.uib.local		2007-11-09 14:39:59
pcbon14-vm0.uib.local	Jans VMWare Test-Client	2007-10-24 13:31:33
pcbon15.uib.local	Testpc Rupert	

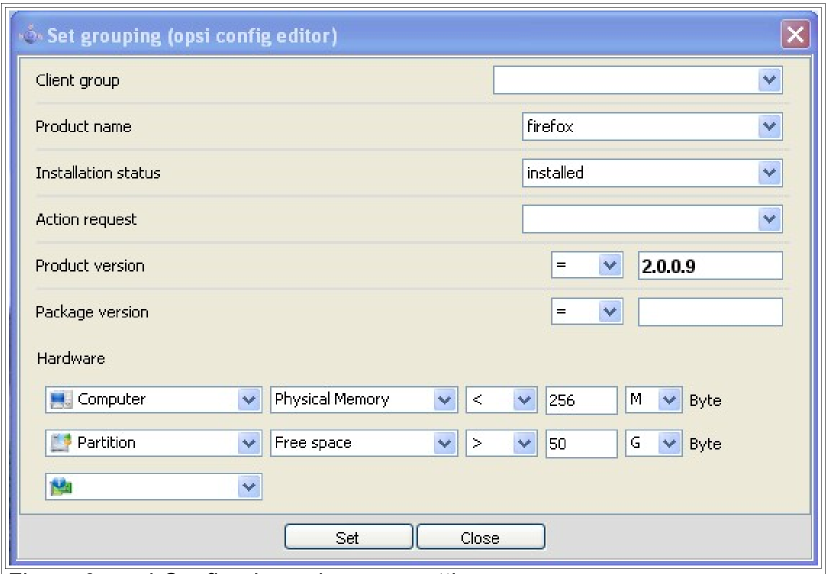
Figure 5: opsi-Configed: client selection mask

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.

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


Set grouping (opsi config editor)

Client group					
Product name	firefox				
Installation status	installed				
Action request					
Product version	=	2.0.0.9			
Package version	=				
Hardware					
Computer	Physical Memory	<	256	M	Byte
Partition	Free space	>	50	G	Byte

Figure 6: opsi-Configed: mask: group setting

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

3.2.4. Client processing / WakeOnLan / Create a Client

You can select one or more clients and send them a 'WakeOnLan' signal by choosing from the menu 'OpsiClient'. There you also have the possibility to create or delete clients.

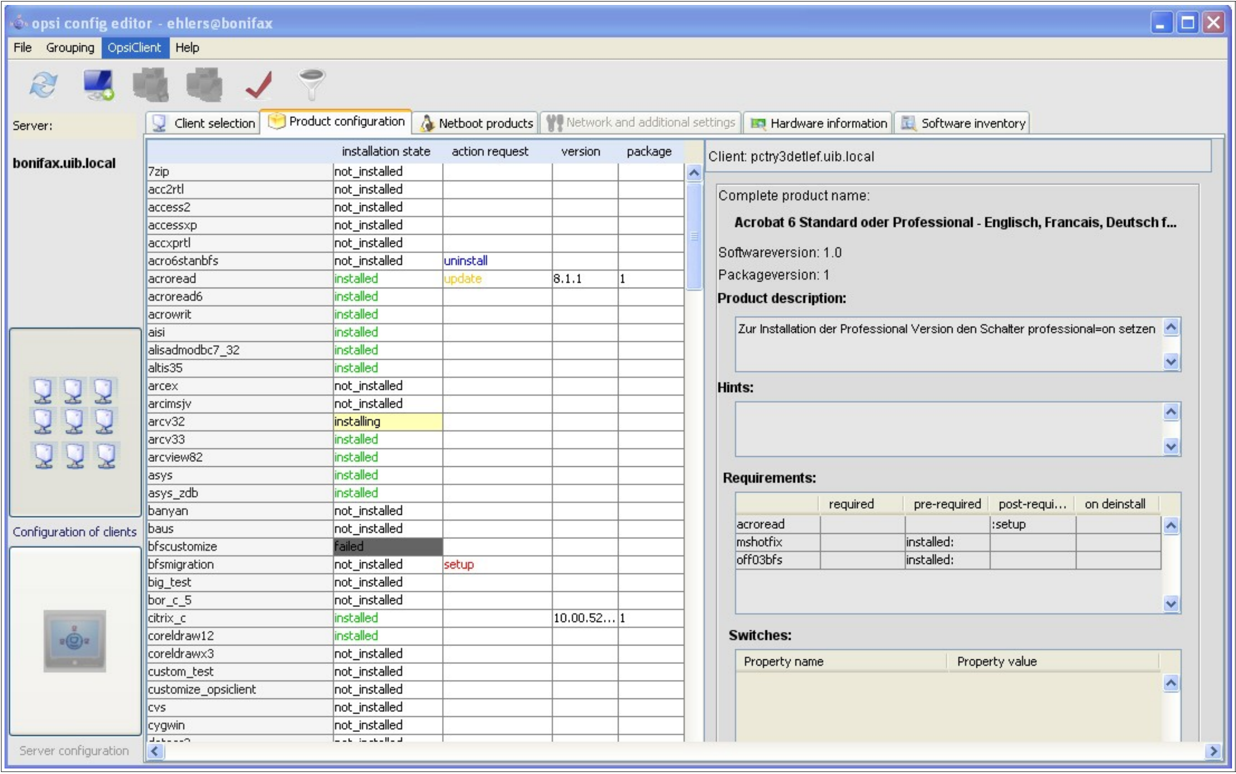
You will find an input mask with the necessary informations for creating a client under the menu point 'Create new opsi client'.

This mask 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.

3.2.5. 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



	installation state	action request	version	package
7zip	not_installed			
acc2rtl	not_installed			
access2	not_installed			
accessxp	not_installed			
accxprtl	not_installed			
acro6stanbfs	not_installed	uninstall		
acroread	installed	update	8.1.1	1
acroread6	installed			
acrowrit	installed			
aisi	installed			
alisadmodbc7_32	installed			
altis35	installed			
arcex	not_installed			
arcimsjv	not_installed			
arcv32	installing			
arcv33	installed			
arcview82	installed			
asys	installed			
asys_zdb	installed			
banyan	not_installed			
baus	not_installed			
bfscustomize	failed			
bfsmigration	not_installed	setup		
big_test	not_installed			
bor_c_5	not_installed			
citrix_c	installed		10.00.52...	1
coreldraw12	installed			
coreldrawx3	not_installed			
custom_test	not_installed			
customize_opsiclient	not_installed			
cvs	not_installed			
cygwin	not_installed			

Client: pctry3detlef.uib.local

Complete product name:

Acrobat 6 Standard oder Professional - Englisch, Francais, Deutsch f...

Softwareversion: 1.0

Packageversion: 1

Product description:

Zur Installation der Professional Version den Schalter professional=on setzen

Hints:

Requirements:

	required	pre-required	post-requi...	on deinstall
acroread			:setup	
mshotfix		installed:		
off03bfs		installed:		

Switches:

Property name	Property value

Figure 7: opsi-Configed: product configuration mask

- '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'

- '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.

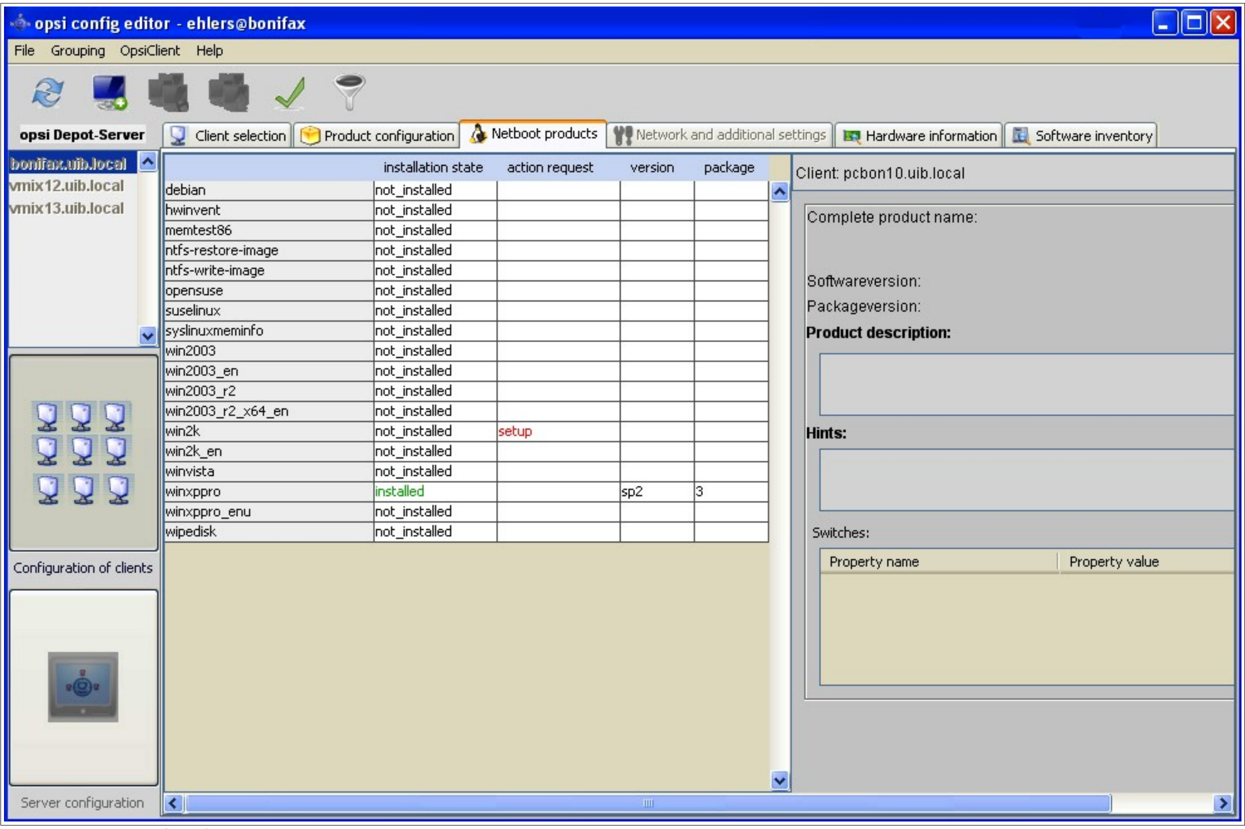
'Switches': For a client specific configuration additional product specific switches can be defined by the product. The list of available switches is shown. The meaning of the switch is shown in the tool tip (when the cursor is moved over the switch name). Under 'property value' you get a list of permitted 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.

3.2.6. 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.)



	installation state	action request	version	package
debian	not_installed			
hwinvent	not_installed			
memtest86	not_installed			
ntfs-restore-image	not_installed			
ntfs-write-image	not_installed			
opensuse	not_installed			
suselinux	not_installed			
syslinuxmeminfo	not_installed			
win2003	not_installed			
win2003_en	not_installed			
win2003_r2	not_installed			
win2003_r2_x64_en	not_installed			
win2k	not_installed	setup		
win2k_en	not_installed			
winvista	not_installed			
winxppro	installed		sp2	3
winxppro_enu	not_installed			
wipedisk	not_installed			

Client: pcbon10.uib.local

Complete product name:

Softwareversion:

Packageversion:

Product description:

Hints:

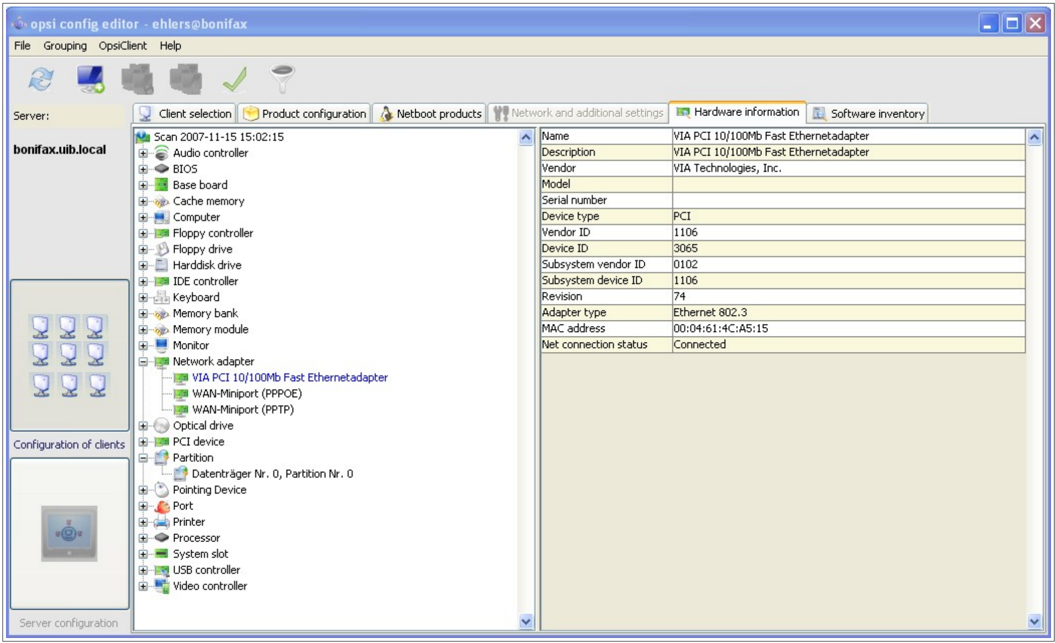
Switches:

Property name	Property value

Figure 8: opsi-Configed: mask to start the bootimage

3.2.7. Hardware information

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

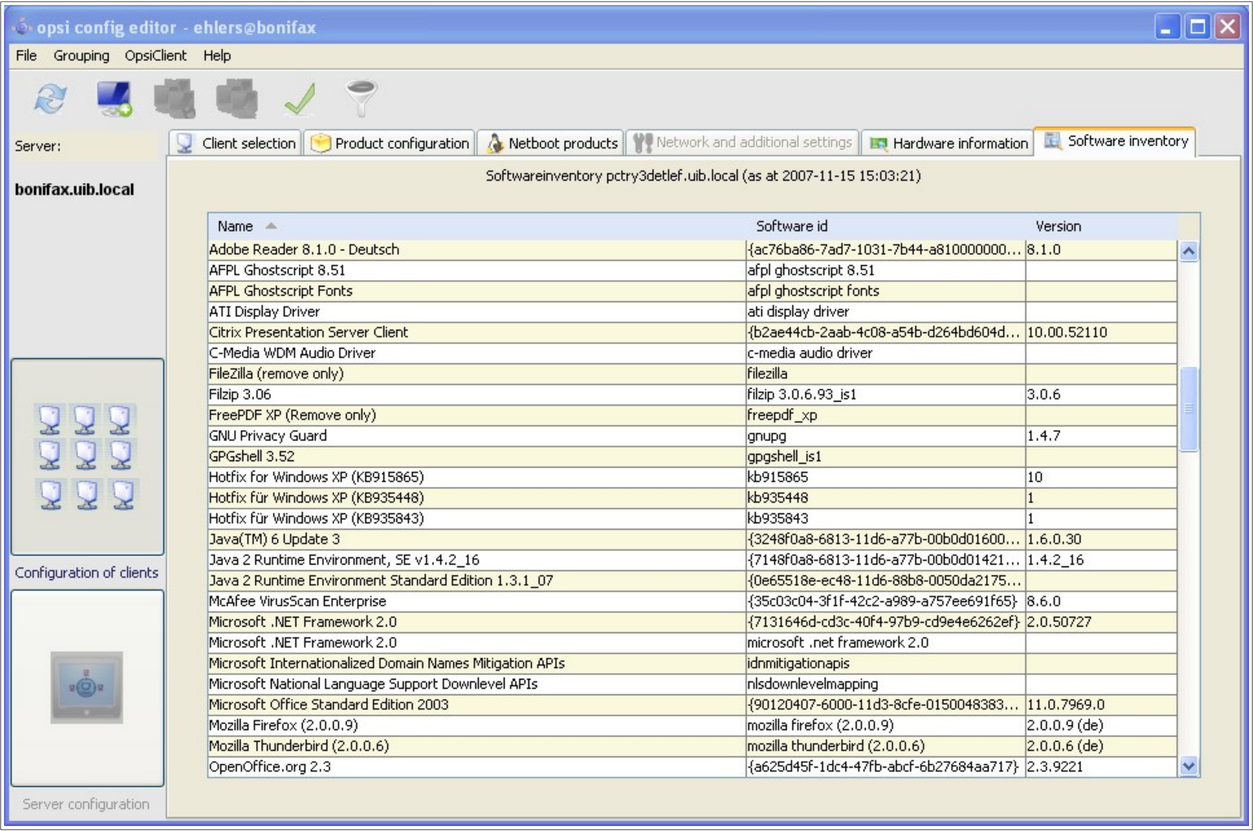


```
Scan 2007-11-15 15:02:15
  Audio controller
  BIOS
  Base board
  Cache memory
  Computer
  Floppy controller
  Floppy drive
  Harddisk drive
  IDE controller
  Keyboard
  Memory bank
  Memory module
  Monitor
  Network adapter
    VIA PCI 10/100Mb Fast Ethernetadapter
    WAN-Miniport (PPPOE)
    WAN-Miniport (PPTP)
  Optical drive
  PCI device
  Partition
    Datenträger Nr. 0, Partition Nr. 0
  Pointing Device
  Port
  Printer
  Processor
  System slot
  USB controller
  Video controller
```

Name	VIA PCI 10/100Mb Fast Ethernetadapter
Description	VIA PCI 10/100Mb Fast Ethernetadapter
Vendor	VIA Technologies, Inc.
Model	
Serial number	
Device type	PCI
Vendor ID	1106
Device ID	3065
Subsystem vendor ID	0102
Subsystem device ID	1106
Revision	74
Adapter type	Ethernet 802.3
MAC address	00:04:61:4C:A5:15
Net connection status	Connected

Figure 9: opsi-Configed: Hardware informations for the selected client

3.2.8. Software inventory



Softwareinventory pctry3detlef.uib.local (as at 2007-11-15 15:03:21)

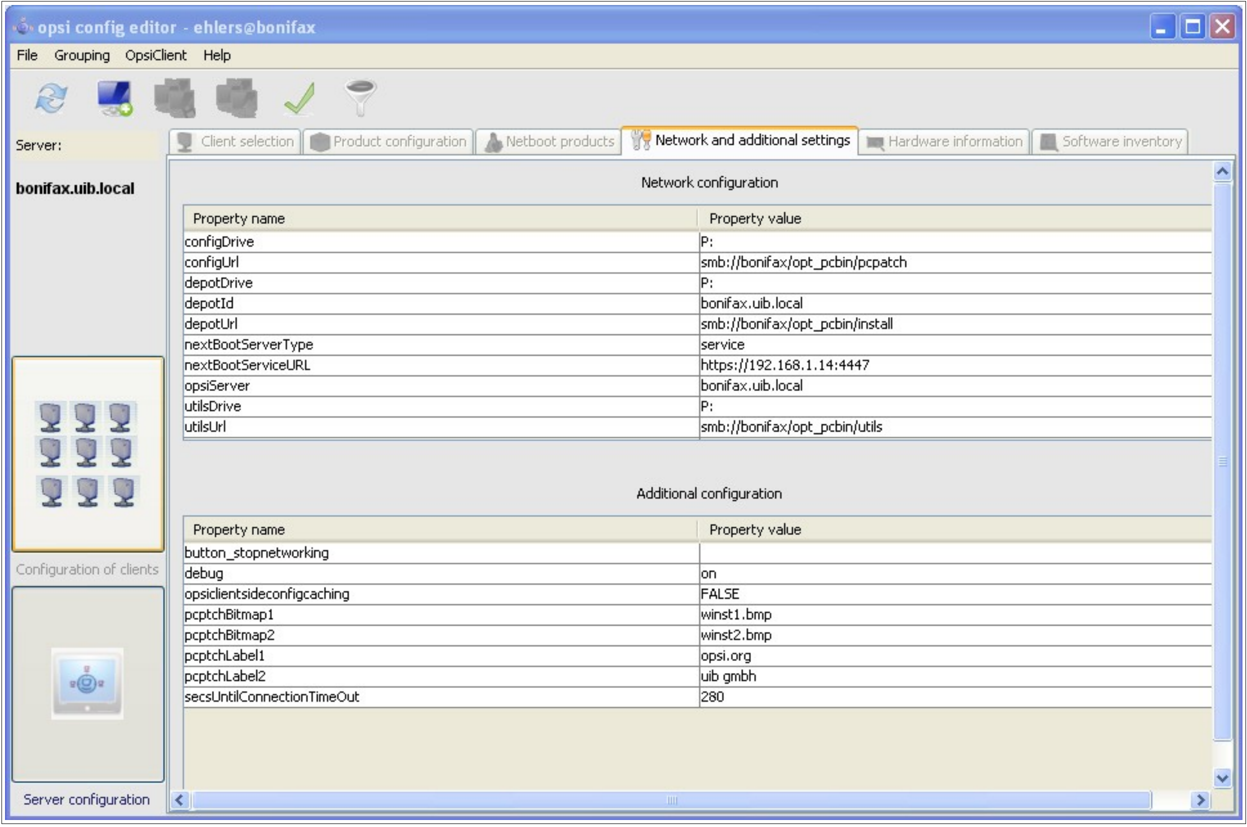
Name	Software id	Version
Adobe Reader 8.1.0 - Deutsch	{ac76ba86-7ad7-1031-7b44-a810000000…	8.1.0
AFPL Ghostscript 8.51	afpl ghostscript 8.51	
AFPL Ghostscript Fonts	afpl ghostscript fonts	
ATI Display Driver	ati display driver	
Citrix Presentation Server Client	{b2ae44cb-2aab-4c08-a54b-d264bd604d…	10.00.52110
C-Media WDM Audio Driver	c-media audio driver	
FileZilla (remove only)	filezilla	
Filzip 3.06	filzip 3.0.6.93_is1	3.0.6
FreePDF XP (Remove only)	freepdf_xp	
GNU Privacy Guard	gnupg	1.4.7
GPGshell 3.52	gpgshell_is1	
Hotfix for Windows XP (KB915865)	kb915865	10
Hotfix für Windows XP (KB935448)	kb935448	1
Hotfix für Windows XP (KB935843)	kb935843	1
Java(TM) 6 Update 3	{3248f0a8-6813-11d6-a77b-00b0d01600…	1.6.0.30
Java 2 Runtime Environment, SE v1.4.2_16	{7148f0a8-6813-11d6-a77b-00b0d01421…	1.4.2_16
Java 2 Runtime Environment Standard Edition 1.3.1_07	{0e65518e-ec48-11d6-88b8-0050da2175…	
McAfee VirusScan Enterprise	{35c03c04-3f1f-42c2-a989-a757ee691f65}	8.6.0
Microsoft .NET Framework 2.0	{7131646d-cd3c-40f4-97b9-cd9e4e6262ef}	2.0.50727
Microsoft .NET Framework 2.0	microsoft .net framework 2.0	
Microsoft Internationalized Domain Names Mitigation APIs	idnmitigationapis	
Microsoft National Language Support Downlevel APIs	nlsdownlevelmapping	
Microsoft Office Standard Edition 2003	{90120407-6000-11d3-8cfe-0150048383…	11.0.7969.0
Mozilla Firefox (2.0.0.9)	mozilla firefox (2.0.0.9)	2.0.0.9 (de)
Mozilla Thunderbird (2.0.0.6)	mozilla thunderbird (2.0.0.6)	2.0.0.6 (de)
OpenOffice.org 2.3	{a625d45f-1dc4-47fb-abcf-6b27684aa717}	2.3.9221

Figure 10: 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).

3.2.9. 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”.



Network configuration

Property name	Property value
configDrive	P:
configUrl	smb://bonifax/opt_pcbin/pcpatch
depotDrive	P:
depotId	bonifax.uib.local
depotUrl	smb://bonifax/opt_pcbin/install
nextBootServerType	service
nextBootServiceURL	https://192.168.1.14:4447
opsiServer	bonifax.uib.local
utilsDrive	P:
utilsUrl	smb://bonifax/opt_pcbin/utils

Additional configuration

Property name	Property value
button_stopnetworking	
debug	on
opsiclientsideconfigcaching	FALSE
pcptchBitmap1	winst1.bmp
pcptchBitmap2	winst2.bmp
pcptchLabel1	opsi.org
pcptchLabel2	uib gmbh
secsUntilConnectionTimeOut	280

Figure 11: opsi-Configed: network and additional configuration

<u>3.3. Tool: opsi V3 opsi-Webconfiged</u>

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 V3 opsi-admin

New in opsi V3.

3.4.1. 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.

3.4.2. Typical use cases

3.4.2.1. Delete product

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

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

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

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

3.4.2.3. Client delete

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

3.4.2.4. Client create

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

3.4.2.5. Client boot image activate

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

3.4.2.6. Attach client description

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

3.4.2.7. Set pcpatch password

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

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

3.4.3. 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 (for example section [general] in *.sysconf file).

`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. Localboot products: automatic software distribution with opsi

4.1. opsi-preloginloader

4.1.1. 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.

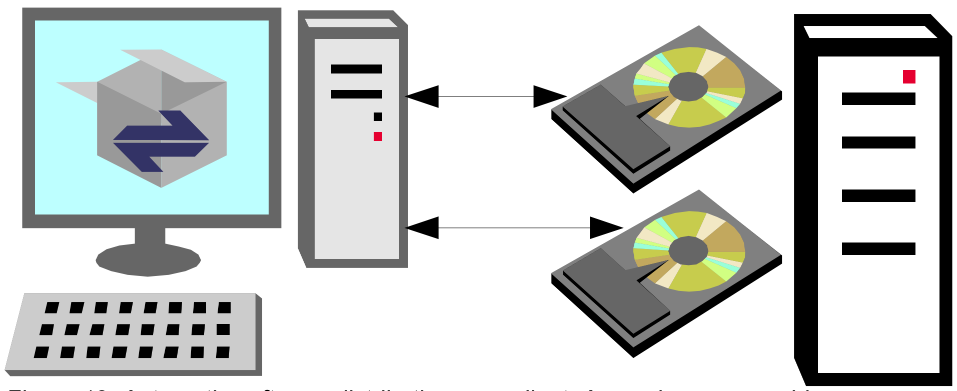


Figure 12: Automatic software distribution on a client. An opsi server provides configuration information and installable software packets.

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. The 'wInst' executable supports the opsi wInst script processing language and and provides different ways of installation:

- Existing setup programs from the original software manufacturer can be executed from within a wInst script in 'silent' or 'unattended' mode. It depends on the setup program whether silent installation mode is supported
- The standard setup can be analyzed and 'recorded' to do the installation tasks directly by the 'wInst' program. Usually that is something like file installation to the local file system and patching the registry
- The interactive answers required by the original setup program can be given automatically by using the free tool 'autoIt' (www.hiddensoft.com/autoit/). That means providing an autoIt script for unattended installation

Usually a combination of all different ways in one script does the job best. Like doing the basic installation by the original setup if available and then do some customizing by patching registry or file based configuration.

4.1.2. Integration of the software installation with the opsi preLoginLoader

The primary objective of software distribution is to accomplish automatic software installation without user interaction. Software installation and user activity should be strictly separated. In most cases the installation process requires administrative privileges which the user usually doesn't have. So the installation process has to be done independently from the user. In that way neither the user can interfere with nor the user is affected by a software installation process.

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.

4.1.3. Subsequent installation of the opsi-preloginloaders

If you would like to integrate an already installed computer in the software distribution system (that means the OS had been installed before), you need to install the 'opsi PreLoginLoader' packet on that computer.

Therefore the pckey (used for password encryption) has to be installed on the client and the server. Therefore you can choose one of the following methods:

4.1.3.1. Usage of the opsi-deploy-preloginloader

The opsi-deploy-preloginloader script installs the opsi-preloginloader direct from the opsi depot server on the clients. Requirements for the clients are:

- an open C$ share
- an open admin$ share

- an administrative account

The script creates the client information on the server and copies the installation files, the configuration information and the pckey to the client and starts the installation on the client.

With the opsi-deploy-preloginloader script you can batch install a list of clients. The script itself is located in /opt/pcbin/install/preloginloader.

Attention: During installation the client reboots without notice!

```
bonifax:/home/uib/oertel# cd /opt/pcbin/install/preloginloader
bonifax:/opt/pcbin/install/preloginloader# ./opsi-deploy-preloginloader -h

Usage: opsi-deploy-preloginloader [options] [host]...
Deploy opsi preloginloader to the specified clients.
The c$ and admin$ must be accessible on every client.
Options:
    -h        show this help text
    -V        show version information
    -v        increase verbosity (can be used multiple times)
    -u        username for authentication (default: Administrator)
    -p        password for authentication
    -f        file containing list of clients (one hostname per line)
```

4.1.3.2. Usage of service_setup.cmd

Also in /opt/pcbin/install/preloginloader located is the script service_setup.cmd. This can be started with administrative privileges from the client side. The script connects to the opsi-webservice to create the server side client information and to get the pckey. The connect takes the user/password combination registered in the config.ini. If the connect fails, a login window pops up to fill in service URL, user and password. The provided user has to be member of the group 'opsiadmin'.

Attention: During installation the client reboots without notice!

4.1.4. Blocking the user login with the opsi-Loginblocker

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

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.

4.2. opsi standard products

4.2.1. opsi-preloginloader

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

4.2.2. opsi-wInst

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

4.2.3. Javavm: Java Runtime Environment

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

4.2.4. opsi-admin

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

4.2.5. 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.

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

4.3.1. Create an opsi-wInst script

4.3.1.1. Overview

For Windows clients there are basically three ways to integrate new software package into the software deployment, and an additional way for using the Microsoft Installer Service (MSI).

1. Unattended / Silent Setup:
   The original manufacturer setup will be run in a non interactive unattended mode (command line arguments for unattended setup).

   The most important special case of this is

2. Running a MSI packet in silent mode:
   An original software manufacturer MSI package is run in silent mode.

3. Interactive setup with automated answers:
   To prepare for this installation mode, first all the windows that pop up during interactive installation have to be analyzed. Then an 'answer file' is created to automatically fill in the answers.

4. Analyze and repackage:
   First it has to be analyzed (semi automated) what data (files, registry ...) get installed

on the client. This could be done by comparing a plain OS installation before and after the installation of the software packet in question. Based on this a new package can be created containing all the new files and registry entries. This package can be created using wInst tools. It can also be transformed into a MSI package to be integrated in any distribution mechanism.

In details:

4.3.1.2. Integration with unattended or silent setup

For an „unattended“ or „silent” setup the original setup will be switched to an unattended non interactive mode by applicable command line arguments.

Once you know the proper command line arguments, the request can be embedded in a wInst-script.

Here an example how to start a setup with the argument '/silent':

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

[Initial]
Message=install xyz ...
StayOnTop=false

[Aktionen]
  Winbatch_produkt_silent_install

[Winbatch_produkt_silent_install]
%SCRIPTPATH%\setup_xyz.exe /silent
```

The command 'stayOnTop=false' is regarding the z-order of visible windows and enables any tasks started from inside the script to set themselves on top of the script window. Usually you will like them to be on top to have setup messages, progress bar and popups of the setup be visible.

To use the unattended setup you would first have to find the command line argument for silent mode, if there is any. Usually it is something like '/s' or '/silent' or '/s /v"/qb-!"' What it is exactly depends on the setup itself, so there is no safe way to get an unattended installation running under all circumstances, but a lot of tips and tricks are available for this.

4.3.1.2.1. Search unattended.sourceforge.net and others

Before you start to integrate a new package, you'd better first have a look at unattended.sourceforge.net whether somebody already did that job:

http://unattended.sourceforge.net/installers.php

http://www.german-nlite.de/index.php?act=module&module=pages&pg=schalterse

There you also can find a lot of hints and switches for most of the current setup programs. And on unattended.sourceforge.net (or uib.de) you can provide your solution other users.

For many software components you will find packages on sourceforge.

For examples have a look at the 'unattended wiki':
http://ubertechnique.com/unattended/Scripts

or cvs.souceforge.net:
http://cvs.sourceforge.net/viewcvs.py/unattended/unattended/install/scripts/

Here for example is described how to install an exe as "msi-packet" or "silent". You can easily transform this into a 'Winbatch' statement.

It also might be helpful to have a look at:

http://www.appdeploy.com/packages/browse.asp?cat=all

http://www.german-nlite.de/index.php?autocom=custom&page=ug-schaltertabelle

http://www.windows-unattended.de/component/option,com_appbase/Itemid,160/

http://www.msfn.org/board/lofiversion/index.php/f80.html

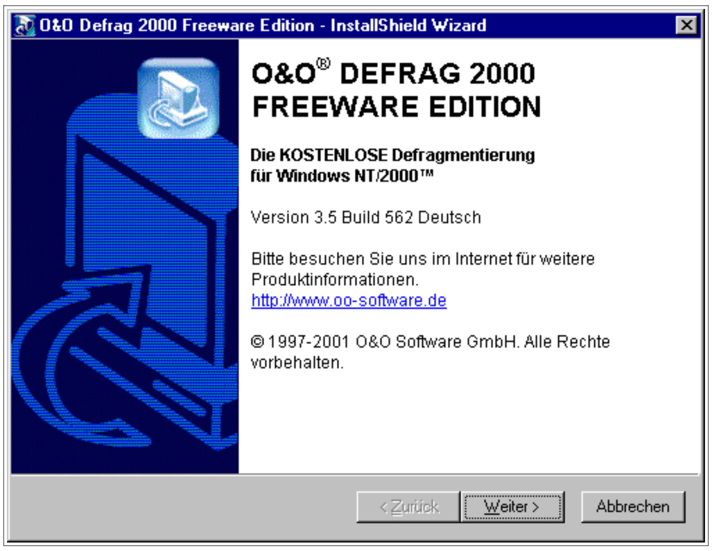
4.3.1.2.2. Search the software producers site

A lot of software manufacturers are aware of the needs of unattended software distribution, so there often are some hints and instructions in the product documentation or on the software producers website.

4.3.1.2.3. Search the setup tool manufacturers site

Often setup programs are not written by the software manufacturers themselves. In most cases they deploy dedicated software products for creating setup programs. So the command line arguments for the resulting packet often are typical for the setup creation tool it is based on.

How to find out what setup tool has been used? Often the name of the manufacturer is displayed in the title of the welcome window. In the following example this is 'Installshield':



O&O Defrag 2000 Freeware Edition - InstallShield Wizard

O&O® DEFRAG 2000
FREEWARE EDITION

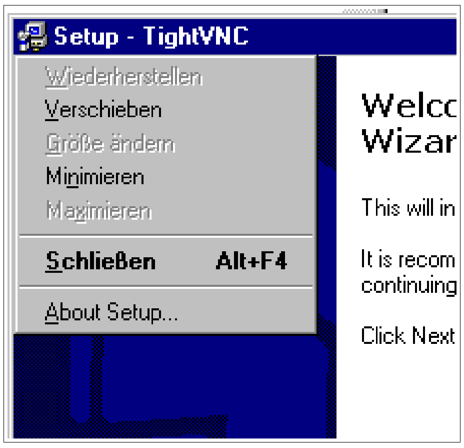
Die KOSTENLOSE Defragmentierung
für Windows NT/2000™

Version 3.5 Build 562 Deutsch

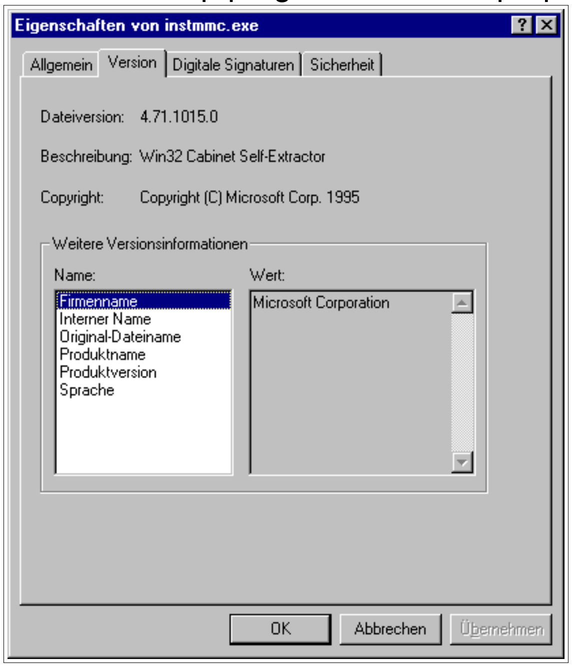
Bitte besuchen Sie uns im Internet für weitere Produktinformationen.
http://www.oo-software.de

© 1997-2001 O&O Software GmbH. Alle Rechte vorbehalten.

Sometimes you find some information in the 'About Setup':



Another source of information is the file version info. You can get this from the file explorer: right mouse click on the setup program and then 'properties'.



Eigenschaften von instmmc.exe

Allgemein | Version | Digitale Signaturen | Sicherheit

Dateiversion: 4.71.1015.0

Beschreibung: Win32 Cabinet Self-Extractor

Copyright: Copyright (C) Microsoft Corp. 1995

Weitere Versionsinformationen

Name: Firmenname, Interner Name, Original-Dateiname, Produktname, Produktversion, Sprache

Wert: Microsoft Corporation

OK | Abbrechen | Übernehmen

On the website of the software manufacturer you can search by the keywords 'silent' , 'silent Install' or 'unattended'.

Here are some manufacturer's links:

http://unattended.sourceforge.net

http://helpnet.installshield.com/robo/projects/InstallShieldXFAQ/FAQDeploymentSilent.htm

http://helpnet.installshield.com/robo/projects/InstallShieldXHelpLib/IHelpSetup_EXECmdLine.htm#sParam

http://helpnet.installshield.com/robo/projects/InstallShieldXHelpLib/SetupIss.htm

http://www.jrsoftware.org/isfaq.php#silent

http://nsis.sourceforge.net/

http://nsis.sourceforge.net/index.php?id=19&backPID=15&tx_faq_faq=39

http://www.wise.com/ (usually /s)

4.3.1.2.4. Installation with a logged on user

As a starting point we assume that you have done an unattended installation by using a wInst-script. The installation works OK when started as a logged on user (with administrative privileges).
But when started from within the software deployment (preloginloader) it fails . A possible reason for that difference might be that the installation process requires an user environment or profile.

In case of a MSI package the option ALLUSERS=2 might help.
Example:

```
[Aktionen]
DefVar $LOG_LOCATION$
Set $LOG_LOCATION$ = "c:\tmp\myproduct.log"
winbatch_install_myproduct

[winbatch_install_myproduct]
msiexec /qb ALLUSERS=2 /l* $LOG_LOCATION$ /i %SCRIPTPATH%\files\myproduct.msi
```

Another more complex way to solve the problem is to create a temporary administrative user account and use this for the program installation. For a detailed description how to do this please refer to the wInst-handbook chapter 8.3 'Script for installation in the context of a local administrator'.

4.3.1.3. Work with MSI-packages

With Windows 2000 Microsoft launched its own installation concept based on the Microsoft Installer Service „MSI“. In the meantime many setup programs are MSI compliant.

To be MSI compliant means to provide a packet with install instructions for the MSI. Usually this is a file named 'product.msi'.

In practice the „setup.exe“ of a product contains a 'product.msi' file and an additional control program for the installation. The control program unpacks the 'product.msi' and pops up a window to ask for the installation's start. If this has been approved the control program checks whether MSI is installed and passes 'product.msi' over. If there is no suitable MSI, the control program first starts the installation of the MSI.

When you interrupt the installation at that point, you often find the unpacked MSI-package in a temporary directory.

This package can be used for unattended installation for instance with the statement:

```
msiexec /qb-! ALLUSERS=2 /i Product.msi
```

or

```
msiexec /i Product.msi /qn
```

Some more particular arguments can be supplied. An overview on the command line arguments of “msiexec.exe” provides:

http://helpnet.installshield.com/robo/projects/InstallShieldXHelpLib/IHelpCmdLineMSI.htm

For more information on MSI have a look at:

http://www.microsoft.com/technet/prodtechnol/windowsserver2003/de/library/ServerHelp/9361d377-9011-4e21-8011-db371fa220ba.mspx

http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/msiexec.mspx

http://www.microsoft.com/windows2000/techinfo/howitworks/management/installer.asp

http://www.microsoft.com/technet/prodtechnol/windows2000serv/maintain/featusability/winmsi.mspx

4.3.1.4. Customizing after a silent/unattended installation

After a successful silent installation some more customizing might be useful. The 'opsi Winst' is a powerful tool to do that job. At first you will have to find out what patches have to be applied. For example that could mean to analyze what registry settings are affected by the GUI customizing tools.

You can use the tools portrayed in chapter 'Analyze and repackage' further down. Some more tools can be found here:

http://www.sysinternals.com/

http://www.german-nlite.de/files/guides/regshot/regshot.html

4.3.1.5. Integration with automated answers for the setup program

Another fast way of integration is to provide an automated answer file for the setup process. To be more precise, the answer file is used by a control tool, which waits for the setup to come up with interactive windows and then passes input to these windows as defined in the answer file. As a control tool we recommend 'AutoIt'. The AutoIt program and the documentation you will find in the opsi-integtools or at the website: http://www.hiddensoft.com/autoit3.

Here is a (very simple) example for using AutoIt:

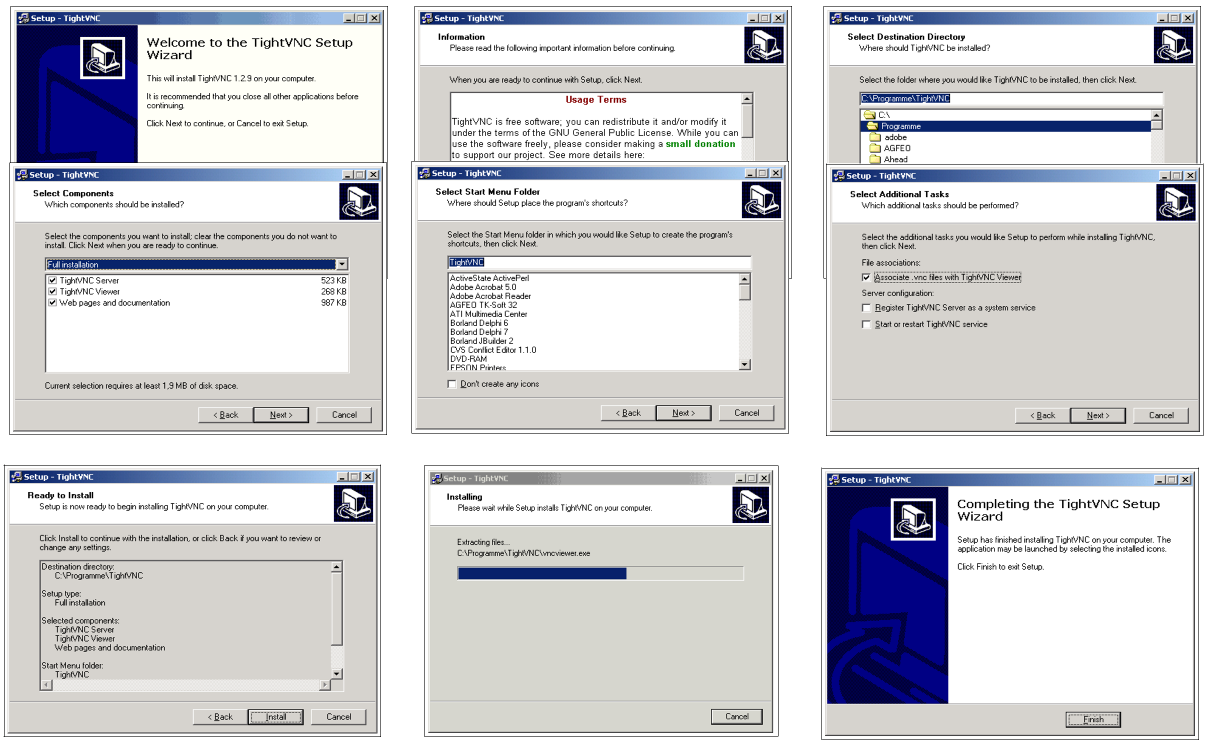
Automated integration of 'TightVNC' with the help of AutoIt.

```
; start the setup
Run, tightvnc-1.2.9-setup.exe
; wait for the first window titled „Setup - TightVNC‟
WinWait, Setup - TightVNC
; Send „Enter‟ to that window:
Send, {ENTER}
; and so on ... send ENTER for default processing:
WinWait, Setup - TightVNC
Send, {ENTER}
WinWait, Setup - TightVNC
Send, {ENTER}
WinWait, Setup - TightVNC
Send, {ENTER}
WinWait, Setup - TightVNC
Send, {ENTER}
WinWait, Setup - TightVNC
Send, {ENTER}
WinWait, Setup - TightVNC
Send, {ENTER}
; at that point the installation has started
WinWait, Setup - TightVNC
; send F for Finish
Send, F
exit
```

At the start of 'autoit.exe' the name of the control file is passed as an argument.

Following screen shots correspond with the answer file. As you can see, sending “Enter” to every window proceeds the default installation:

AutoIt provides a lot of commands to control the setup process. Also several error states can be handled (if known in advance) with the [ADLIB] section in the script.



Welcome to the TightVNC Setup Wizard

This will install TightVNC 1.2.9 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Information
Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

Usage Terms

TightVNC is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License. While you can use the software freely, please consider making a small donation to support our project. See more details here:

Select Destination Directory
Where should TightVNC be installed?

Select the folder where you would like TightVNC to be installed, then click Next.

C:\Programme\TightVNC

C:\
Programme
adobe
AGFEO
Ahead

Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

☑ TightVNC Server	523 KB
☑ TightVNC Viewer	268 KB
☑ Web pages and documentation	987 KB

Current selection requires at least 1,9 MB of disk space.

Select Start Menu Folder
Where should Setup place the program's shortcuts?

Select the Start Menu folder in which you would like Setup to create the program's shortcuts, then click Next.

TightVNC

ActiveState ActivePerl
Adobe Acrobat 5.0
Adobe Acrobat Reader
AGFEO TK-Soft 32
ATI Multimedia Center
Borland Delphi 6
Borland Delphi 7
Borland JBuilder 2
CVS Conflict Editor 1.1.0
DVD-RAM
EPSON Printers

☐ Don't create any icons

Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing TightVNC, then click Next.

File associations:
☑ Associate .vnc files with TightVNC Viewer

Server configuration:
☐ Register TightVNC Server as a system service
☐ Start or restart TightVNC service

Ready to Install
Setup is now ready to begin installing TightVNC on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

Destination directory:
    C:\Programme\TightVNC

Setup type:
    Full installation

Selected components:
    TightVNC Server
    TightVNC Viewer
    Web pages and documentation

Start Menu folder:
    TightVNC

Installing
Please wait while Setup installs TightVNC on your computer.

Extracting files...
C:\Programme\TightVNC\vncviewer.exe

Completing the TightVNC Setup Wizard

Setup has finished installing TightVNC on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

Although there is a fundamental challenge in using AutoIt:
The AutoIt-Script must provide input for every window, that might pop up during installation. So if any unexpected window pops up, which isn't handled in the [ADLIB] section, AutoIt provides no input for this window and the installation stops at that point waiting for input. This input could be done by an interactive user and then the script can take over again and handles the next well known windows.

There is another critical path of an AutoIt-Installation:
The user can interfere with the installation if the mouse and keyboard are not disabled. Therefore we regard 'unattended' or 'silent' setup as a more stable solution.

A combination of both might do a good job:
The 'silent'-setup does the main installation and the AutoIt-script handles special conditions that might occur.

Example: wInst-script for installation of 'TightVNC'

```
[Initial]
Message=install tightvnc 1.2.9 ......

[Action]
; start AutoIt as background process to absorb message windows,
; which appear during the tightvnc installation run as service
winbatch_tightvnc_autoit_confirm  /LetThemGo
; start the setup as a silent setup
winbatch_tightvnc_silent_install

[winbatch_tightvnc_autoit_confirm]
%SCRIPTPATH%\autoit %SCRIPTPATH%\confirm.aut

[winbatch_tightvnc_silent_install]
%SCRIPTPATH%\tightvnc-1.2.9-setup.exe /silent
```

4.3.1.6. Analyze and repackage

When a software developer builds a setup for deployment, he usually knows about the required components of the software that have to be installed. But if somebody just has got the setup as a black box, he first needs to analyze what the setup does. This can be done by monitoring the setup activities with appropriate tools (e.g. monitoring any file and registry access) or by comparing the system states before and after installation.

To analyze the before / after states, Microsoft provides the utility sysdiff for Windows NT and Windows 2000. This program can perform a system snapshot before and after the setup process. Also it comes up with options to compare both states, lists and collects the differing components and even builds a simple installation script from that. Such a script could be the base for a reproduction of the original installation.

But the sysdiff installation routines and packages aren't compatible with the new MSI-standard. Probably due to this reason sysdiff is no longer supported for Windows XP. As a replacement for this Microsoft server edition comes with a tool for packaging of MSI-packages – the product **WinINSTALL LE** from the company OnDemand.

WinINSTALL LE used to be available as freeware. In the meantime just a trial version can be obtained from:

http://www.ondemandsoftware.com

On Windows XP WinINSTALL LE basically does the same as sysdiff does on Windows NT / 2000. A system snapshot before and after a test installation leads to building a MSI package containing all the differences. This package should be valid for installation with the Microsoft Installer.

Microsoft also provides the tool Orca to inspect MSI packages. Basically (if the job wasn't too complex) Orca can edit and create MSI packages and also checks existing MSI packages for integrity. Orca is part of the platform SDK for Windows Server 2003, which can be obtained from:

http://www.microsoft.com/msdownload/platformsdk/sdkupdate/

Another interesting free tool for creation of MSI packages is the program Installer2GO from the company Dev4PC. Here comes the link:

http://dev4pc.com

In the following you will get some tips for working with WinINSTALL LE and Orca.

4.3.1.6.1. Hints for execution of WinINSTALL LE

At first WinINSTALL LE has to be installed on a server running the same operating system as the target client, usually Windows XP.

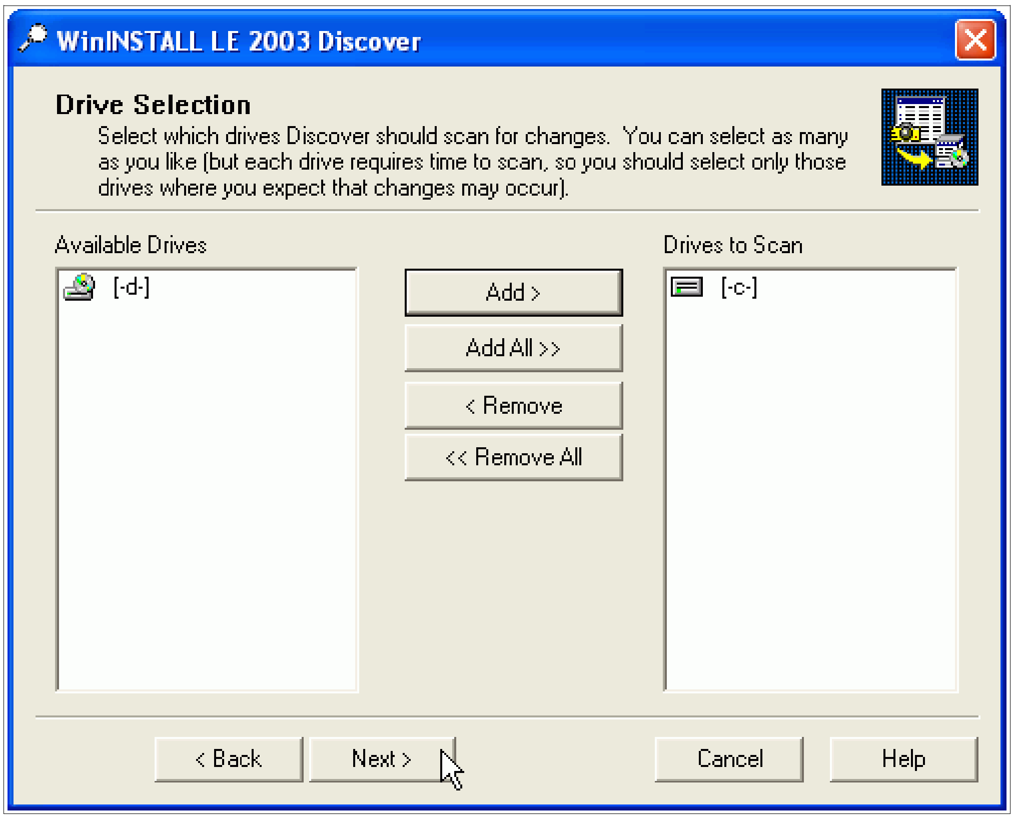
For a test installation of the software to be analyzed, you need a recently installed target client with only the required software components installed – usually just the system software including updates.

During installation of WinINSTALL LE on the server, the setup programm asks for a file share to save the installation packets to and creates the share if it doesn't already exist.

The procedure of analysis and creating the MSI package will be started from the target client side. First connect to the server share and start the discover program ('disco32.exe'). It will guide you through the further steps.

The first step is to name the application, e.g. “program 1.0” and configure the path for the package. Every package should have its own folder.

The discover program prepares the snapshot of the original system state. Therefore it will ask you what drives should be scanned during snapshot. Normally only the system and program files drive (C:\) has to be monitored, for setup programs usually do not touch other drives.


WinINSTALL LE 2003 Discover

Drive Selection

Select which drives Discover should scan for changes. You can select as many as you like (but each drive requires time to scan, so you should select only those drives where you expect that changes may occur).

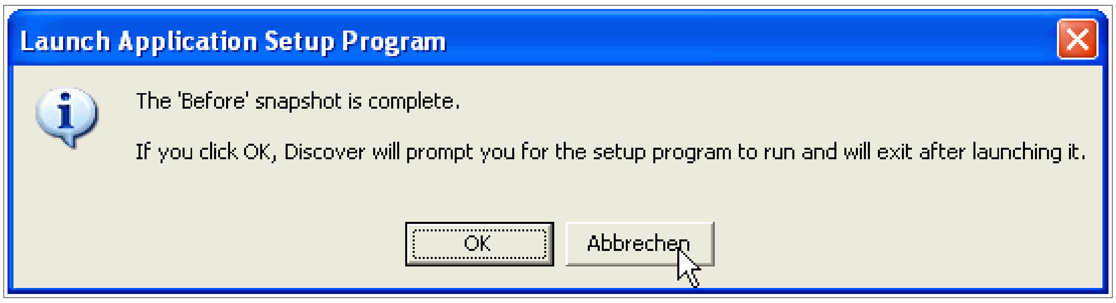
Available Drives: [-d-]

Drives to Scan: [-c-]

Furthermore the discover program presents a list of subdirectories, registry sections and configuration files to be excluded from the analysis, for they are e.g. temporary files or contain no relevant information. The list can be edited, however you better use the default setting if there is no good reason for a change and you don't have enough background knowledge.

Then the capture of the actual system state (first snapshot) can be started.

The accomplishment of the first snapshot needs some time, and when done, the discover program sends a message to start the original setup of the application:



Launch Application Setup Program

The 'Before' snapshot is complete.

If you click OK, Discover will prompt you for the setup program to run and will exit after launching it.

Sometimes a setup program wants to perform a reboot after installation. You will have to decide case by case whether to do so or not. Microsoft makes no explicit statement how to handle this. When the setup is done, 'disco32.exe' can be started again to take the second snapshot and analyze the differences between the two snapshots. Then it creates a MSI package for software distribution. This package contains everything to reproduce the installation.

4.3.1.6.2. Orca

Microsoft outlines the functions of the program “Orca” as follows:

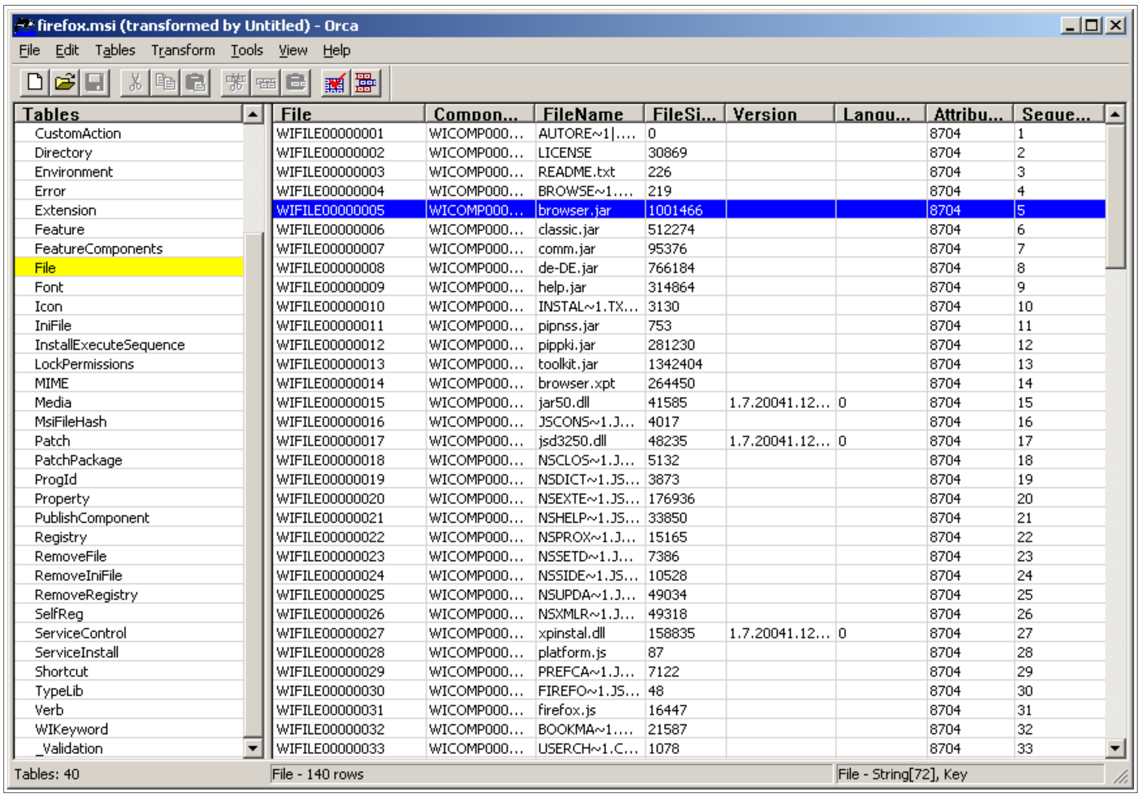
Because of the restrictions of existing tools for the Windows Installer, it might be necessary to edit the MSI packages. To perform this task, Orca is provided by the Windows Installer SDK.

In principle with Orca you can do a lot of jobs concerning MSI packages, but for more complex tasks it is too difficult to handle. So Ocra is suitable mainly for performing some small patches and customization.

Anyway Orca is a great tool to discover the internal structure of MSI packages and to understand the basic principles of a setup information database.

After opening a MSI package with Orca a list of tables will be shown that look like database tables. And that is what they basically are.

When you click on the table named “Files” it will look like this:


firefox.msi (transformed by Untitled) - Orca
File Edit Tables Transform Tools View Help

Tables
CustomAction
Directory
Environment
Error
Extension
Feature
FeatureComponents
File
Font
Icon
IniFile
InstallExecuteSequence
LockPermissions
MIME
Media
MsiFileHash
Patch
PatchPackage
ProgId
Property
PublishComponent
Registry
RemoveFile
RemoveIniFile
RemoveRegistry
SelfReg
ServiceControl
ServiceInstall
Shortcut
TypeLib
Verb
WIKeyword
_Validation

File	Compon...	FileName	FileSi...	Version	Langu...	Attribu...	Seque...
WIFILE00000001	WICOMP000...	AUTORE~1|....	0			8704	1
WIFILE00000002	WICOMP000...	LICENSE	30869			8704	2
WIFILE00000003	WICOMP000...	README.txt	226			8704	3
WIFILE00000004	WICOMP000...	BROWSE~1....	219			8704	4
WIFILE00000005	WICOMP000...	browser.jar	1001466			8704	5
WIFILE00000006	WICOMP000...	classic.jar	512274			8704	6
WIFILE00000007	WICOMP000...	comm.jar	95376			8704	7
WIFILE00000008	WICOMP000...	de-DE.jar	766184			8704	8
WIFILE00000009	WICOMP000...	help.jar	314864			8704	9
WIFILE00000010	WICOMP000...	INSTAL~1.TX...	3130			8704	10
WIFILE00000011	WICOMP000...	pipnss.jar	753			8704	11
WIFILE00000012	WICOMP000...	pippki.jar	281230			8704	12
WIFILE00000013	WICOMP000...	toolkit.jar	1342404			8704	13
WIFILE00000014	WICOMP000...	browser.xpt	264450			8704	14
WIFILE00000015	WICOMP000...	jar50.dll	41585	1.7.20041.12...	0	8704	15
WIFILE00000016	WICOMP000...	JSCONS~1.J...	4017			8704	16
WIFILE00000017	WICOMP000...	jsd3250.dll	48235	1.7.20041.12...	0	8704	17
WIFILE00000018	WICOMP000...	NSCLOS~1.J...	5132			8704	18
WIFILE00000019	WICOMP000...	NSDICT~1.JS...	3873			8704	19
WIFILE00000020	WICOMP000...	NSEXTE~1.JS...	176936			8704	20
WIFILE00000021	WICOMP000...	NSHELP~1.JS...	33850			8704	21
WIFILE00000022	WICOMP000...	NSPROX~1.J...	15165			8704	22
WIFILE00000023	WICOMP000...	NSSETD~1.J...	7386			8704	23
WIFILE00000024	WICOMP000...	NSSIDE~1.JS...	10528			8704	24
WIFILE00000025	WICOMP000...	NSUPDA~1.J...	49034			8704	25
WIFILE00000026	WICOMP000...	NSXMLR~1.J...	49318			8704	26
WIFILE00000027	WICOMP000...	xpinstal.dll	158835	1.7.20041.12...	0	8704	27
WIFILE00000028	WICOMP000...	platform.js	87			8704	28
WIFILE00000029	WICOMP000...	PREFCA~1.J...	7122			8704	29
WIFILE00000030	WICOMP000...	FIREFO~1.JS...	48			8704	30
WIFILE00000031	WICOMP000...	firefox.js	16447			8704	31
WIFILE00000032	WICOMP000...	BOOKMA~1....	21587			8704	32
WIFILE00000033	WICOMP000...	USERCH~1.C...	1078			8704	33

Tables: 40 File - 140 rows File - String[72], Key

On the right side there is a list of all files required for installation. Each file has got an unique key for identification, which is shown in the first column.

In a similar way registry entries and other components are managed.

Another interesting table is the special table „InstallExecuteSequence". The column "Sequence" holds the order of installation. Sorting the table by this column shows the chronological order of installation steps.

4.3.1.7. Internal structure of an integrated product

According to opsi any commercial or free software will be integrated as a product (an installable opsi product). Based on the results of the foregoing software analysis, the opsi product contains all the files for installation and a special install script which will be executed by the opsi installer **wInst**.

4.3.1.7.1. Tasks of the opsi installer wInst (for Windows)

In addition to the Windows installer (MSI) or other commercial installers, the opsi installer **wInst** has to perform following tasks:

- The opsi installer **wInst** provides seamless integration with the opsi configuration data base, which holds the specifications for the product installation.
- Based on **wInst** the installation becomes configurable, customizable and adaptable down to any detail as required for the overall concept.
- **wInst** features detailed logging of all software installation steps which can be very useful for integration and support. The log level can be set gradually from verbose log down to none.

4.3.1.7.2. General hints for writing a Winst-script

4.3.1.7.2.1. What if the installation needs a reboot

Many software components installed by a setup program require a reboot after installation. The reason for this is that performing a reboot often is the only way to make sure, that new configurations and replaced modules come into operation.

So opsi **wInst** comes with a script command for performing a reboot. In the [action] section of the **wInst-**script the command **ExitWindows** can be set and specified with the option **/RebootWanted**, **/Reboot** or **/ImmediateReboot**:

- **/RebootWanted** is the “weakest” reboot option. The reboot request is noted down in the registry and will be performed after all the installations are done. Which means that **wInst** might continue installing other products before rebooting. So reboot requests from many packets can be combined in a final single reboot, instead of rebooting after each packet.
- **/Reboot** launches a reboot after completing the current installation script.
- **/ImmediateReboot** aborts the script execution and performs an immediate reboot. The script itself has to take care for status handling and proper continuation. To control script processing, the script might write a status flag before reboot. After

reboot the first task is to evaluate the status flag and continue with the next part of the installation process. The flag handling might look like this:

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

if not ($RebootFlag = "1")
     ;=========================
     ;
     ; instructions before reboot ...
     ; ... any actions to perform before reboot
     Set $RebootFlag = "1"
     Registry_SaveRebootFlag
     ExitWindows /ImmediateReboot

else
     ;=========================
     ;
     ; instructions after reboot ...
     ; set back rebootflag
     Set $RebootFlag = "0"
     Registry_SaveRebootFlag
     ; ... any actions to perform after reboot

endif
```

In addition there is a **wInst** section required for handling the status flag:

```
[Actions]
[Registry_SaveRebootFlag]
openKey [$WinstRegKey]
set "RebootFlag" = "$RebootFlag"
```

4.3.1.7.2.2. Files copy

The available copy commands are described in the **wInst** handbook.

4.3.1.7.2.3. Start menu entries

In the **wInst** script the start menu entries can be set by a link folder section and the following request:

```
[LinkFolder_adminutils]
set_basefolder common_programs
set_subfolder "Admin Utils"

set_link
  name: Ini-Editor
  target: javaw.exe
  parameters:  -jar %ProgramFilesDir%\opsi.org\inied\inied.jar
  working_dir: $TEMP$
  icon_file: %ProgramFilesDir%\opsi.org\inied\config_prog.ico
  icon_index: 0
end_link

set_link
  name: WinMerge
  target: %ProgramFilesDir%\WinMerge\WinMerge.exe
  parameters:
  working_dir: $TEMP$
  icon_file: %ProgramFilesDir%\WinMerge\WinMerge.exe
  icon_index: 0
end_link
```

4.3.1.7.2.4. System software dependencies

At first in the **wInst** script you have to declare for which system software (or groups of system software) the script is valid. At any point the script can detect the current OS and perform different tasks according to the OS.

In the [action] section of the **wInst** script the system software can be detected by calling the function **GetOS**. The return value of **GetOS** is one of the following:

```
"Windows_16"
"Windows_95"
"Windows_NT"
```

- return value = “Windows_95” for Win95, Win98 and WinME
- return value = “Windows_NT” for Windows NT 4.0, Windows 2000 and Windows XP

If the return value is “Windows_NT”, the minor version can be retrieved by calling the function **GetNTVersion**. One of the following values will be returned:

```
"NT4"
"Win2k"
"WinXP"
"Win NT 5.2" (e.g. Windows Server 2003 R2 Enterprise Edition)
```

A single script can handle different operating systems by using 'if' statements:

```
DefVar  $OS$
set $OS$ = GetOS

DefVar $MinorOS$
set $MinorOS$ = GetNTVersion


; ... perform any tasks for all of the Windows versions
; case switches
if  $OS = "Windows_NT"
      if $MinorOS$ = "NT4"
            ; ... tasks to perform for WinNT only
      else
            ; ... tasks to perform for Win2000/XP
      endif
else
      ; tasks for Win95 family
endif
```

If a script is valid only for the Windows_NT family (Windows NT, Windows 2000 or Windows XP), the following statement prevents from performing the installation on any other OS:

```
if GetOS = "Windows_NT"
; ... if OS=Windows then perform the tasks
endif
; perform no tasks for other OS
```

4.3.1.7.2.5. Options in the wInst script

For some products it might be useful to provide several installation options. So the Internet Explorer packet might be installed only to perform a system software update. But on some PCs also the browser functionality of the Internet Explorer should be installed. To distinct several installation options, a switch can be set in the opsi data base, which is specific for the opsi packet and the individual computer. Using the file backend, the switch is set in the INI-file of the PC and can be evaluated by the function IniVar (flag name) like this:

```
if IniVar ("ie6_exe") = "on"
      Files_CopyC_hiddenexefiles
      Registry "%SCRIPTPATH%\hiddenexe.rgm"
      Registry "%SCRIPTPATH%\hiddenexe.rgu" /AllNTUserDats
endif
```

So if the flag “ie6_exe”=”on”, all files will be copied and registry entries will be set for installing IE as a browser.

The number of switches for any packet is not limited. So for instance this virscan package comes with three different switches:

```
[virscan-install]
....
WriteScan=on
NetworkScan=on
```

```
Reinst451=on
```

It is up to the script how to handle different switch settings. In this case some script variables are set according to the switches, which are used for patching the configuration file 'vsconfig.ini' later on:

```
; patch configuration files and import them
....
if IniVar ("ReadScan") = "off"
  Set $ReadScan="0"
else
  Set $ReadScan="1"
endif
if IniVar ("WriteScan") = "off"
  Set $WriteScan="0"
else
  Set $WriteScan="1"
endif
if IniVar ("NetworkScan") = "off"
  Set $NetworkScan="0"
else
  Set $NetworkScan="1"
endif
....
```

4.3.1.8. How to deinstall products

To deinstall a software product from a computer, you need an 'uninstall' script to perform the deletion. The fundamental difficulty in software deletion is to distinguish what exactly has to be removed. Not all of the files that came with a software package can be deleted afterwards. Sometimes a packet comes with standard modules, which are also referred to by other programs. Often only the software manufacturer himself knows what parts have to be removed. The manufacturer's setup might offer an unattended deinstall option which can be embedded in the opsi deinstall script. Otherwise wInst provides several commands for software deletion:

4.3.1.8.1. Using an uninstall routine

If the product manufacturer provides an option for software deletion, it has to be checked whether it can be run unattended (in silent mode). If it requires some user interaction, an autoIt-script combined with the uninstall routine might do the job. The uninstall statement can be embedded in a [winbatch] section of the wInst-script:

```
[Winbatch_start_ThunderbirdUninstall]
%SYSTEMROOT%\UninstallThunderbird.exe /ma
```

When using an uninstall program, it always should be tested whether all of the files have been deleted and the computer is still in a stable state.

Products which are installed by MSI often come also with an uninstall option, which usually is the **`msiexec.exe`** parameter **`/x`**. And the parameter **`/qb-!`** is for unattended mode (without user interaction). So this is the statement for unattended deinstall:

```
msiexec.exe /x myPacket.msi /qb-!
```

Instead of the package name you could also use the GUID (Global Unique ID) with **`msiexec.exe`**. This GUID identifies the product in the system and can be found in the registry directory

```
HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall
```

A request using the GUID looks like this:

```
msiexec.exe /x {003C5074-EB37-4A75-AC4B-F5394E08B4DD} /qb-!
```

If none of these methods is available or sufficient, the deinstallation can be done by a wInst-script as described in the following:

4.3.1.8.2. Useful wInst commands for uninstall

If a product has been installed by wInst functions, or if there is no uninstall routine for the product, the complete deinstallation has to be done by a wInst script. WInst comes with some powerful uninstall functions. In this chapter we will have an overview, for detailed information refer to the wInst handbook.

The base of deletion is deleting one or more files from the file system. This command can be executed from a wInst files section:

```
delete -f filename
```

or to delete a directory including sub directories:

```
delete -sf dirname
```

The parameter 'f' means 'force' – to delete the files anyway, even if they are marked as 'read only' – and the parameter 's' means including the 'subdirectories'. A file or directory can be deleted from all user profiles by using the option '/AllNTUserProfiles' (see wInst-handbook for details).

Directories containing files with the attribute 'hidden' or 'system' can be deleted by using a 'DosInAnIcon'-section:

```
[DosInAnIcon_deleteDir]
rmdir /S /Q "<dirname>"
```

To stop a running process before deletion use the **killtask** command with the process' name (look at the task manager for process name)**:**

```
killtask "thunderbird.exe"
```

If the product – or part of it – runs as a service, you will have to stop the service before deleting the files. One way to do so, is to set the service to state “inactive“ in the registry and restart the computer. Or to stop the service by using the command **'net stop'**, which doesn't need a reboot:

```
net stop <service name>
```

Also deleting DLL files requires special attention, since DLLs could also be used by other products. There is no general concept for handling this.

To delete registry entries with the wInst you can use the command **`DeleteVar`**. This command deletes entries from the currently open key:

```
DeleteVar <VarName>
```

To delete a registry key with all sub keys and registry variables, you can use the wInst command **DeleteKey**:

```
DeleteKey [HKLM\Software\Macromedia]
```

4.3.2. Creating an opsi package

opsi has a package format which contains the installation files, the opsi wInst installation script and meta data.

The essential advantages of this format are:

- Simplified menu driven handling with the program 'opsi-newprod'.
- Holding all meta data in one file which is easy to edit.
- Optional menu driven install of the package with optional default overriding.
- Information about the package including product version, package version and customer extensions will be saved. The package information is stored in the installation directory and are to be seen in the package name and the opsi-configeditor. In this way different package versions can be handled easily (product life cycle management).
- For creating and unpacking products no root privileges are required. Privileges of the group 'pcpatch' are sufficient.

The packet itself is merely a Gzip compressed cpio archive. This archive includes three directories:

CLIENT_DATA

holds the files which are to be copied into the product directory (/opt/pcbin/install/<productid>).

SERVER_DATA

Holds directories which will be unpacked to /. Root privileges for unpacking might be required.

OPSI

The file named 'control' holds the product meta data (like the product dependencies). The files 'preinst' and 'postinst' will be executed before and after the installation. Any customer extensions might be added here.

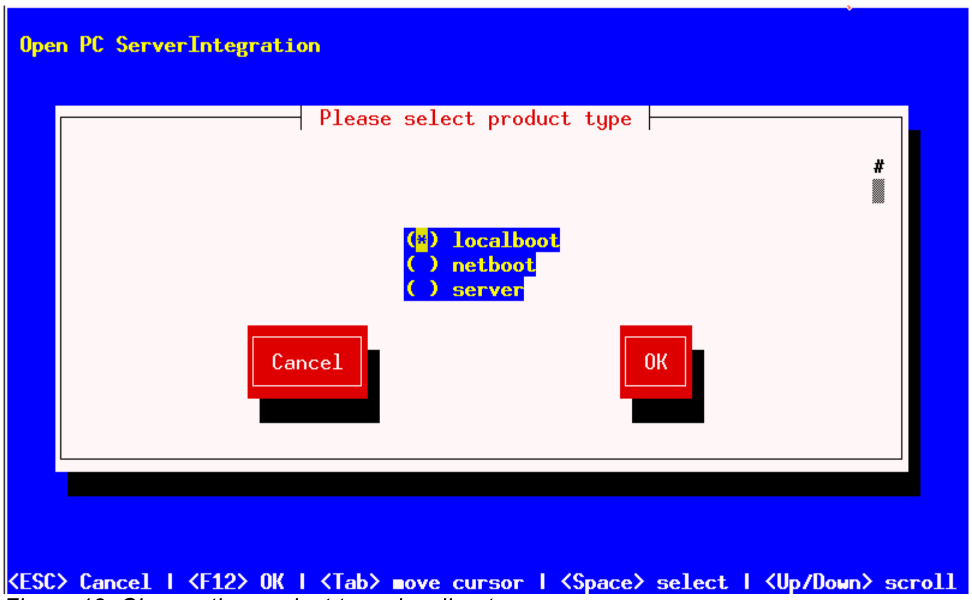
<u>4.3.2.1. Create, pack and unpack a new product</u>

The privileges of the group 'pcpatch' are required to create a new product.

In this example the product will be created in the directory '/home/opsiproducts'. The group 'pcpatch' has to be owner of the directory and the directory permissions are 2770 ('set group ID' bit is set for group pcpatch).

Attention: Do not use any country-specific symbols (umlaut), since the actual country code might vary for different code tables.

Change directory to the product directory and start the creation of the new product with 'newprod'. The next question is for the type of product to create. Choose type 'localboot' for products which should be installable by preloginloader/wInst. Product type 'netboot' is used for products which are activated as a bootimage (like hardware inventory) and type 'server' is used for products that are just installed on a server.

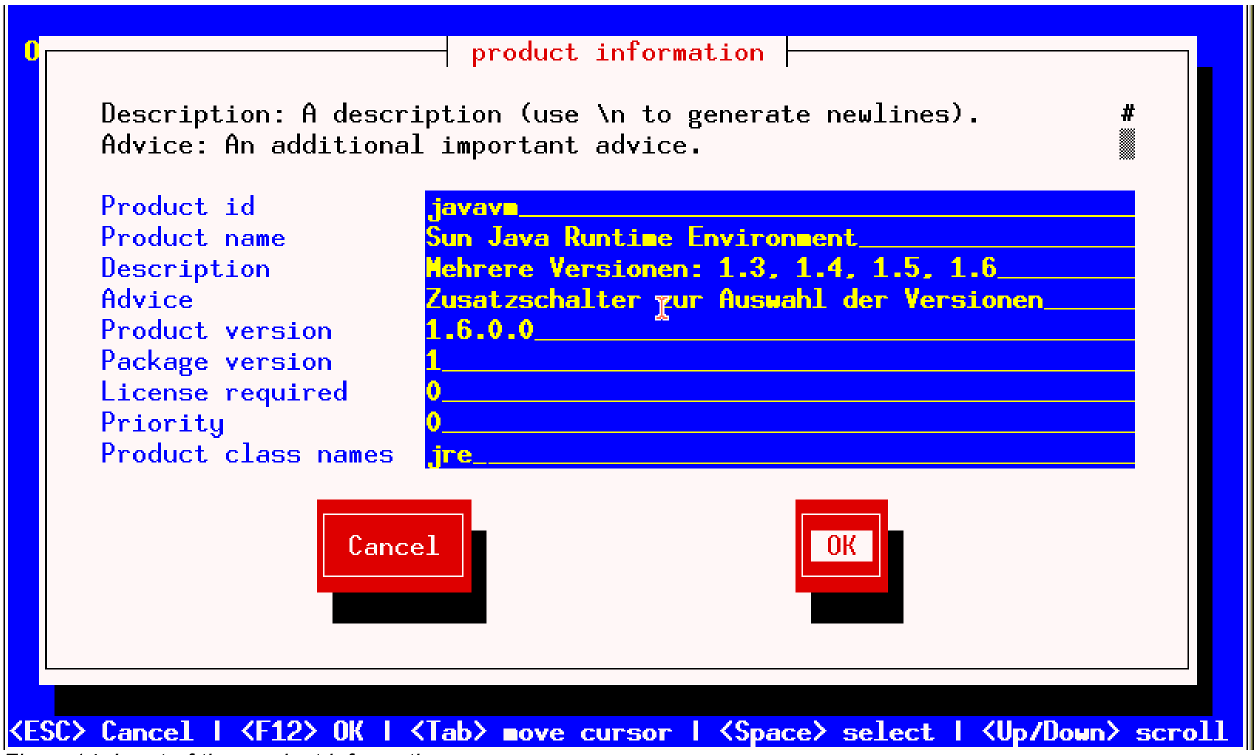


```
Open PC ServerIntegration

Please select product type

(*) localboot
( ) netboot
( ) server

Cancel        OK

<ESC> Cancel | <F12> OK | <Tab> move cursor | <Space> select | <Up/Down> scroll
```

Figure 13: Choose the product type: localboot

Confirm your choice with tab (or F12). Next fill in the basic product parameters. At the top of the window is an explanation for the current input field.

- 'Product id' is a distinct short name for the product, independent from the product version (in opsi V2 this has been the product name)



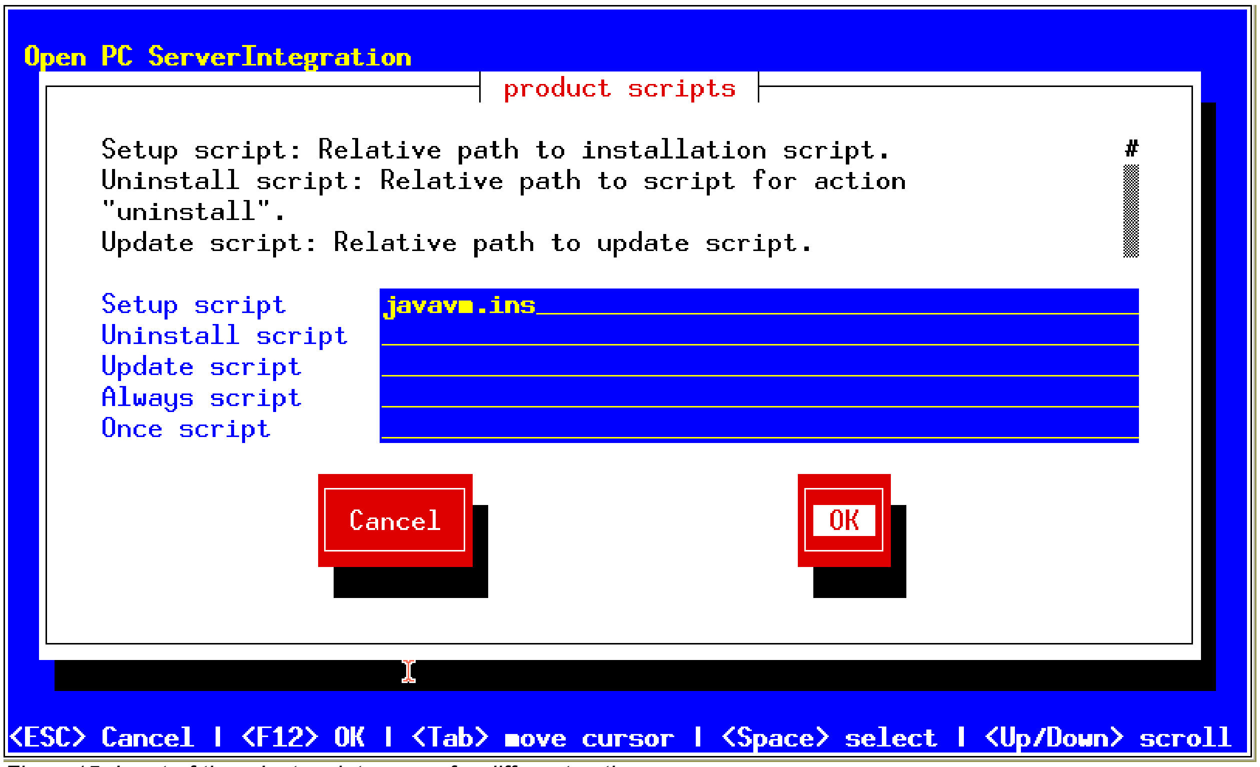
```
                    product information

Description: A description (use \n to generate newlines).        #
Advice: An additional important advice.

Product id             javavm
Product name           Sun Java Runtime Environment
Description            Mehrere Versionen: 1.3, 1.4, 1.5, 1.6
Advice                 Zusatzschalter zur Auswahl der Versionen
Product version        1.6.0.0
Package version        1
License required       0
Priority               0
Product class names    jre

              Cancel                       OK

<ESC> Cancel | <F12> OK | <Tab> move cursor | <Space> select | <Up/Down> scroll
```

Figure14: Input of the product information

- 'Product name' is the full name of the product
- 'Description' is an additional description of the product.
- 'Advice' is some additional information how to handle the product (a note).
- 'Product version' is the version of the packed software.
- 'Package Version' is the version of the package for the product version. This helps to differ packages with the same product version but with for instance a modified wInst script.
- 'Priority' is for future use (regarding the installation order).
- 'Product class names' is for future use.

After the product information is completed, fill in which action scripts should be provided:

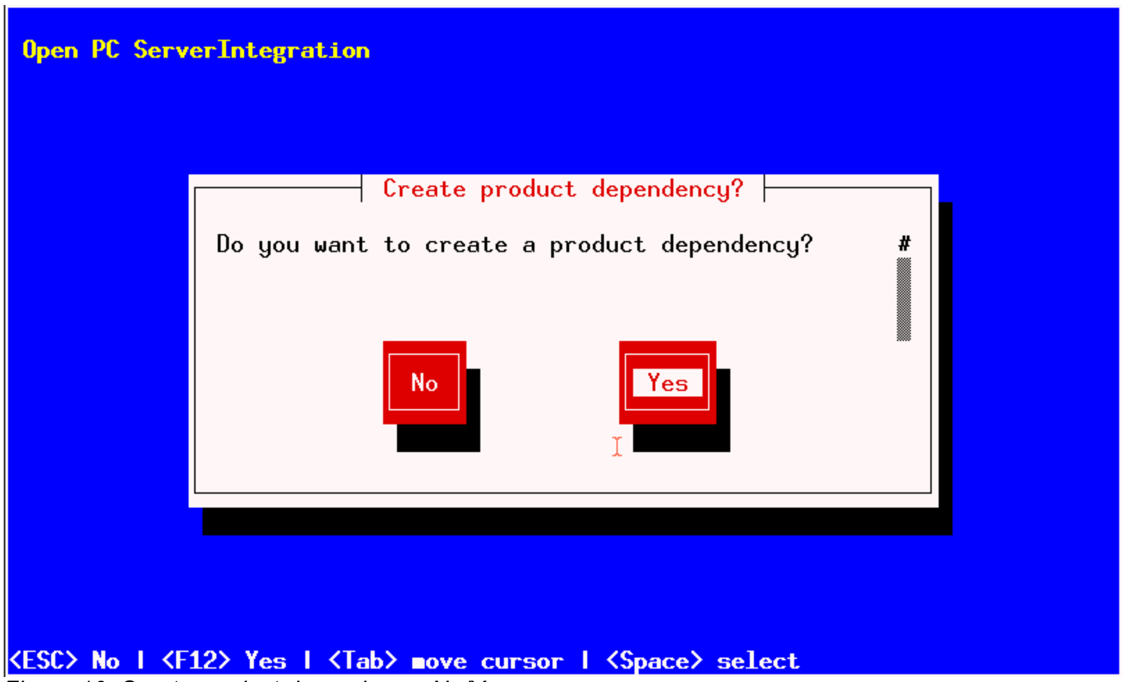
Usually the 'Setup script' is named <productId>.ins.



```
Open PC ServerIntegration
                         product scripts

   Setup script: Relative path to installation script.              #
   Uninstall script: Relative path to script for action
   "uninstall".
   Update script: Relative path to update script.

   Setup script        javavm.ins
   Uninstall script
   Update script
   Always script
   Once script

              Cancel                          OK

<ESC> Cancel | <F12> OK | <Tab> move cursor | <Space> select | <Up/Down> scroll
```

Figure15: Input of the wInst script names for different actions

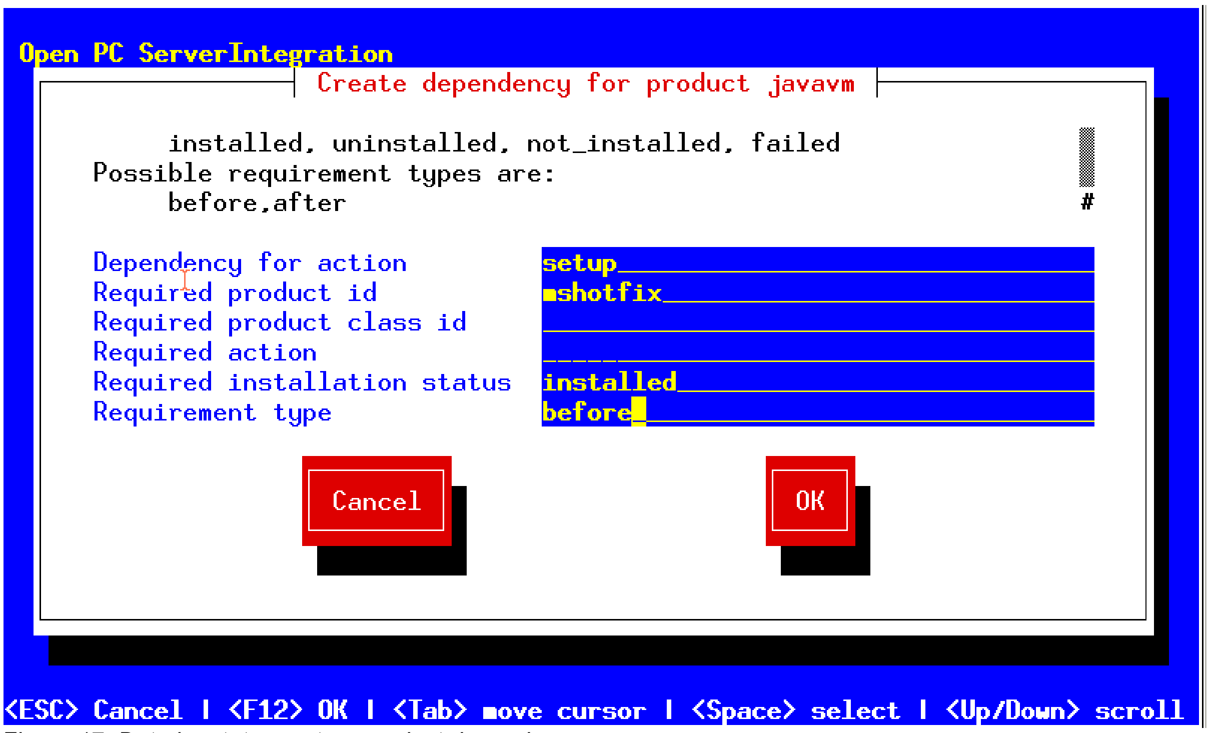
The next step is to define one or more product dependencies. If there are no product dependencies put in 'No'.



```
Open PC ServerIntegration

             Create product dependency?
   Do you want to create a product dependency?        #

                 No            Yes

<ESC> No | <F12> Yes | <Tab> move cursor | <Space> select
```

Figure 16: Create product dependency: No/Yes

To create a product dependency put in the following data (help is available at the top of the window):



```
Open PC ServerIntegration
              Create dependency for product javavm

        installed, uninstalled, not_installed, failed
    Possible requirement types are:
        before,after                                          #

    Dependency for action              setup
    Required product id                shotfix
    Required product class id
    Required action
    Required installation status       installed
    Requirement type                   before

                    Cancel                         OK

<ESC> Cancel | <F12> OK | <Tab> move cursor | <Space> select | <Up/Down> scroll
```

Figure 17: Data input to create a product dependency

'Dependency for action' : For which product action shall the dependency be created (setup, deinstall,...)

- 'Required product id': Product id of the required product

'Required product class id': For future use, leave it empty!

- 'Required action': Select the required action (if any) for the required product. Actions can be as 'setup', 'deinstall', 'update'. If no 'required action' is set, a 'required installation status' must be set

- 'Required installation status': Select the required status of the required product (if any). Usually this is 'installed'. So the required product will be installed if it isn't installed on the client yet. If no 'required installation status' is set, a 'required action' must be set

- 'Requirement type': This is regarding the installation order. If the required product has to be installed **before** the installation of the actual product, this is set to 'before'. If it has to be installed **after** the actual product, set 'requirement type' to 'after'. Leave it blank if the installation order doesn't matter.

After defining a product dependency you will be asked whether to create another product dependency. If you choose 'Yes', the procedure for defining a product dependency is repeated; if you choose 'No' you will be asked to define some product properties, which means defining additional switches for product customization.

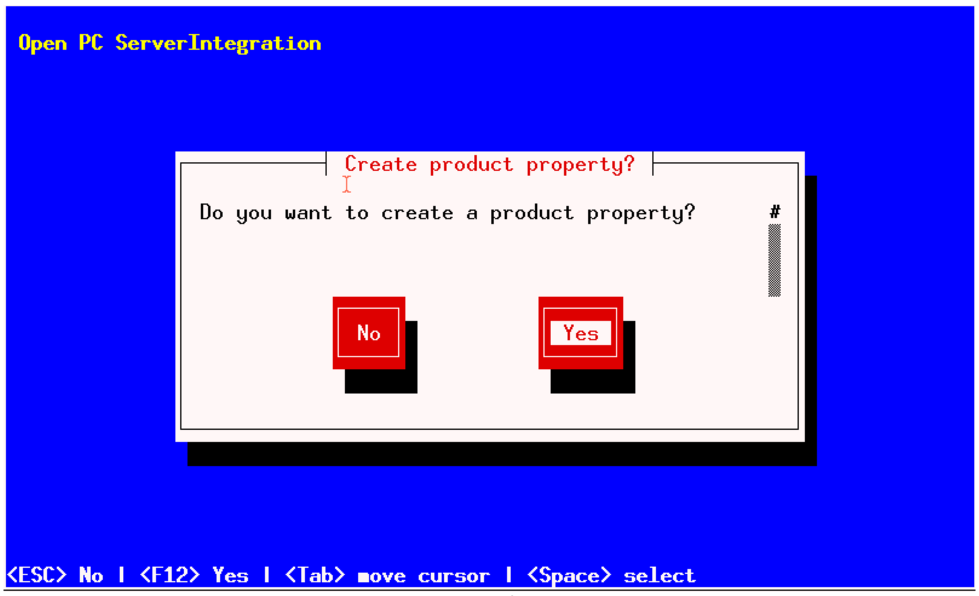
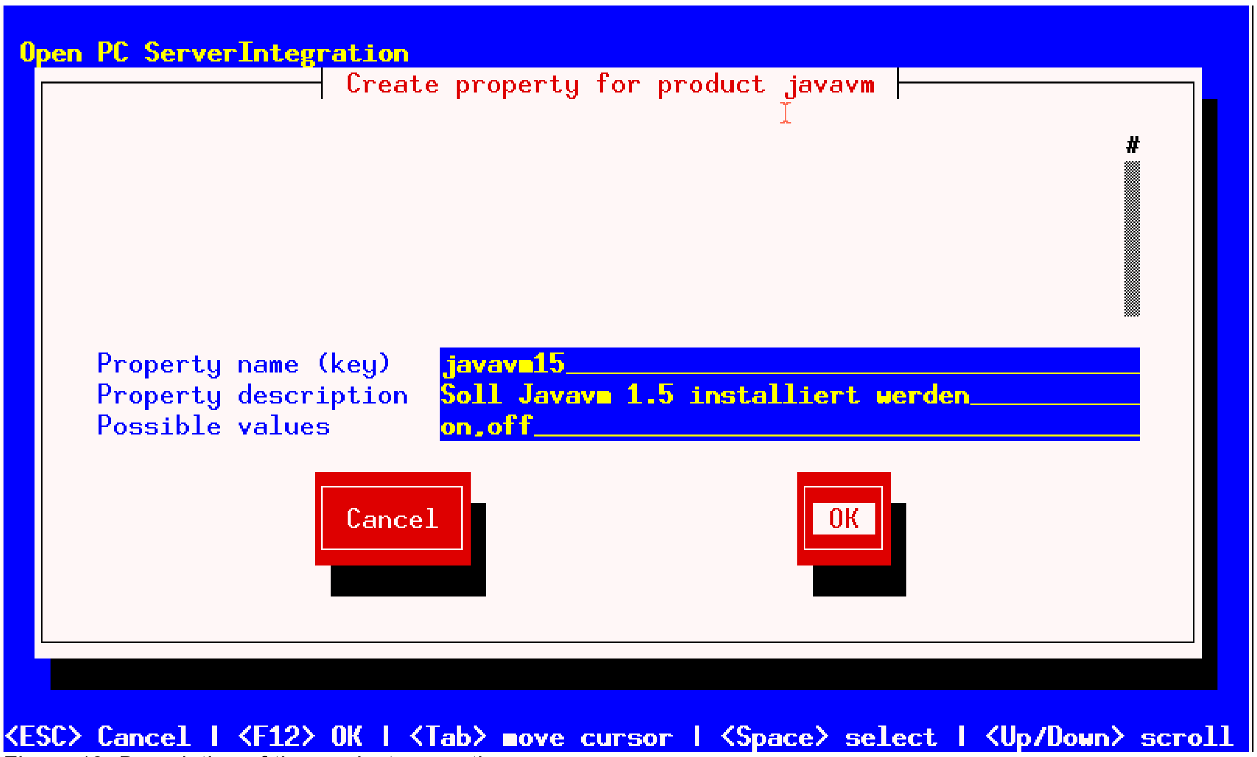

```
Open PC ServerIntegration

Create product property?
Do you want to create a product property?

No    Yes

<ESC> No | <F12> Yes | <Tab> move cursor | <Space> select
```

Figure 18: A(nother) product property to create?

If you answer 'Yes' you will have to describe the product properties.

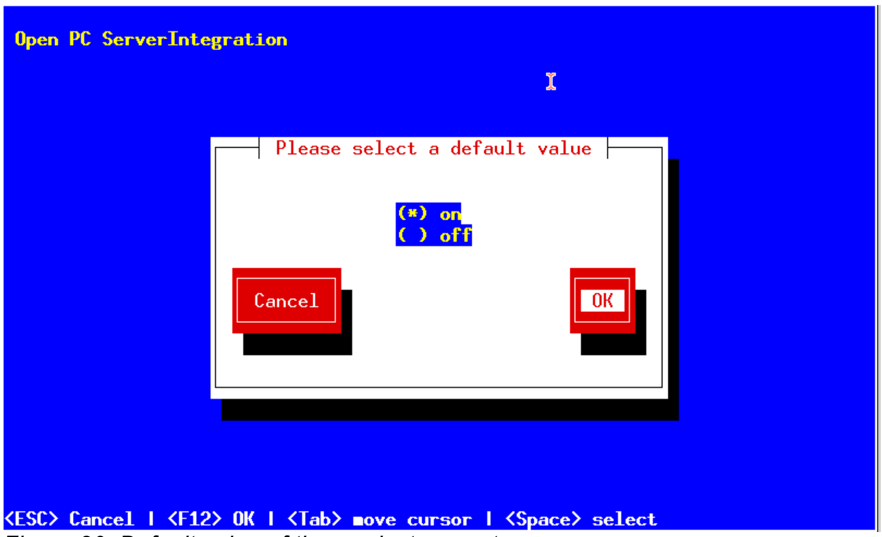
The product properties are client specific and are a name (key) which can hold different values. These values can be evaluated by the wInst-script and result in installing different options at installation time. Further on a description for the switch hneeds to be specified, which will be shown in the opsi-configeditor as a help text and also when the package is unpacked. Next you can define the set of values for the switch (separated by comma). If this is left blank, any value is allowed for the switch.



```
Open PC ServerIntegration
              Create property for product javavm

   Property name (key)     javavm15
   Property description    Soll Javavm 1.5 installiert werden
   Possible values         on,off

          Cancel                        OK

<ESC> Cancel | <F12> OK | <Tab> move cursor | <Space> select | <Up/Down> scroll
```

Figure 19: Description of the product properties

In the following you can define the default value of the product property (switch).



```
Open PC ServerIntegration

          Please select a default value

                 (*) on
                 ( ) off

     Cancel                        OK

<ESC> Cancel | <F12> OK | <Tab> move cursor | <Space> select
```

Figure 20: Default value of the product property

After defining a product property, you will be asked whether to create another product property. If you choose 'Yes' the procedure of defining a property repeats; if you choose 'No', the basic definitions for the new product are done.

Using the list command (ls) you can see the directory structure as described above. Change to the OPSI folder and list the content. The 'control' file now contains the data you just have defined and you can load the file into an editor to view or change the entries.

Example of a 'control' file:

```
[Product]
type: localboot
id: javavm
name: Sun Java Runtime Environment
description:
Mehrere Versionen: 1.3, 1.4, 1.5, 1.6
advice: Additional choice switch for different versions
version: 1.6.0.0
packageVersion: 1
priority: 0
licenseRequired: True
productClasses: jre
setupScript: javavm.ins
uninstallScript:
updateScript:
alwaysScript:
onceScript:

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

[ProductProperty]
name: default13
description: on=Version 1.3 are default JRE; off=The actual installed Javavm
are the default JRE
values: on, off
default: off

[ProductProperty]
name: javavm15
description: Should Javavm 1.5 be installed
values: on, off
default: on

[ProductProperty]
name: javavm16
description: Should Javavm 1.6 be installed
values: on, off
default: off
```

As the next step you will have to copy the product wInst-script and the necessary data files into the CLIENT_DATA folder.

Then you can pack the package. Change to the root directory of the product and start 'makeproductfile'. The product will be packed.

'makeproductfile' can be started with different options:

```
# makeproductfile -h

Usage: makeproductfile [-h] [-v|-s] [-f] [-l log-level] [-i custom name]
[source directory]
Provides an opsi package from a package source directory.
If no source directory is supplied, the current directory will be used.
Options:
   -v          verbose
   -s          silent
   -l          log-level 0..6
   -f          fast, no topicality test
   -i          custom name
```

The packet then can be installed on any opsi server with the command 'opsiinst <packet name>'.

Also 'opsiinst' has some optional parameters:

```
# opsiinst -h

Usage: opsiinst [options] <opsi package>
Install an opsi package.
Options:
    -v                show version information
    -h                this help text
    -q                quiet, don't ask any questions
    -f                force installation
    -d                enable debug mode
    -l <log-file>     use a file for logging
    -i <interface>    user interface [snack|text]
```

Without additional parameters 'opsiinst' will start in interactive modus, so you can e.g. modify the default product properties for your server. If you would like to install a product without any questions, start 'opsiinst' with the options '-q -f'.

4.3.2.2. Create client specific opsi packages

Sometimes a product needs to have different installation options for different servers, maybe regarding the license agreement or because of different customization options. To handle this, you will have to save the specific data files in separate subdirectories before packing the package. When packing, the -i option **`makeproductfile -i`** selects which of the custom subdirectories will be included into the packet.

Example:
The product 'softprod' shall be created in three versions. There is a base version and two other versions with additional software, one for 'CompanyX' and the other for 'CustomerY'. Therefore in the directory '**`/home/opsiproduct/softprod/`**' we create the folders '**`CLIENT_DATA.CompanyX`**' and '**`CLIENT_DATA.CustomerY`**'. In both the new directories you create another folder '**`custom_ins_dir`**' which contains the *additional* software of the respective version. Later on, when installing the package on a server, the custom folder '**`custom_ins_dir`**' will become a direct subdirectory of the main product folder:

`/home/opsiproduct/softprod/custom_ins_dir`

It is also possible to create another wInst-script in this directory, which can be called from the actual setup wInst-script during client installation. The custom wInst-script can be started from the main wInst-script like this:

```
if FileExists("%ScriptPath%\custom_ins_dir\custom.ins")
  sub "%ScriptPath%\custom_ins_dir\custom.ins"
endif
```

To pack the product package you will have to change to the base product directory '**`/home/opsiproduct/softprod/`**'. Then pack the product for 'CompanyX' by calling '**`makeproductfile -i CompanyX`**'. Use the same name as the corresponding **`CLIENT_DATA`**-directory !
So: **`CLIENT_DATA.<custom name>`** => **`makeproductfile -i <custom name>.`**
The request for packing the package for 'CustomerY' is done similar:
'**`makeproductfile -i CustomerY`**'

The base version of the package (without any custom extension) will be packed by just calling 'makeproductfile' without -i parameter.

The custom package
`softprod.<product version>-<packet version>_<custom name>.opsi`
can be distributed and installed like any other packet.

5. Netboot products: Automated OS installation and more

5.1. Unattended automated OS installation

5.1.1. 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.

5.1.2. 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.

5.1.3. 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.

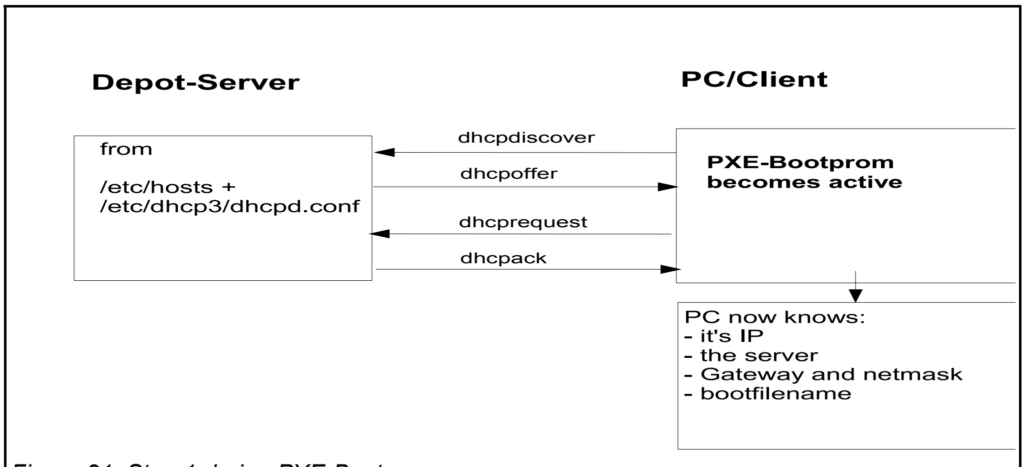


Figure 21: Step 1 during PXE-Boot

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.

5.1.3.1. 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.

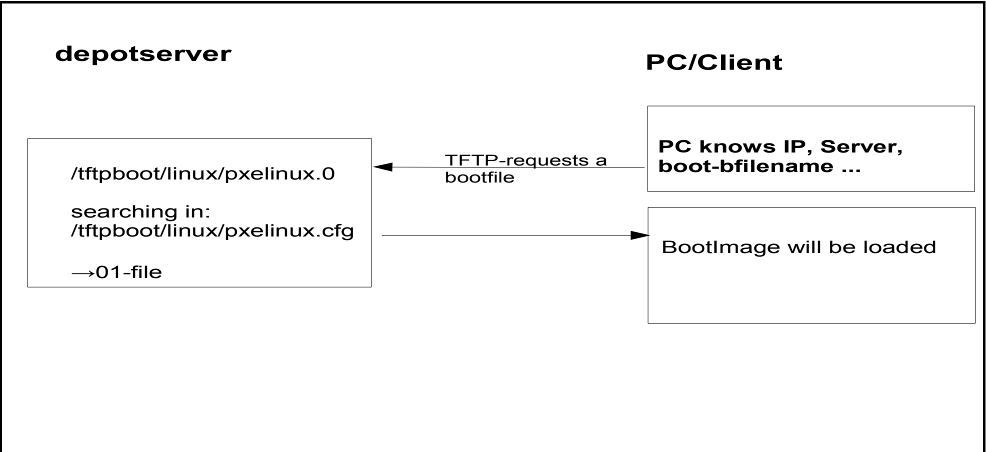

depotserver
PC/Client
/tftpboot/linux/pxelinux.0
searching in:
/tftpboot/linux/pxelinux.cfg
→01-file
TFTP-requests a bootfile
PC knows IP, Server, boot-bfilename ...
BootImage will be loaded

Figure 22: 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.

5.1.4. 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.

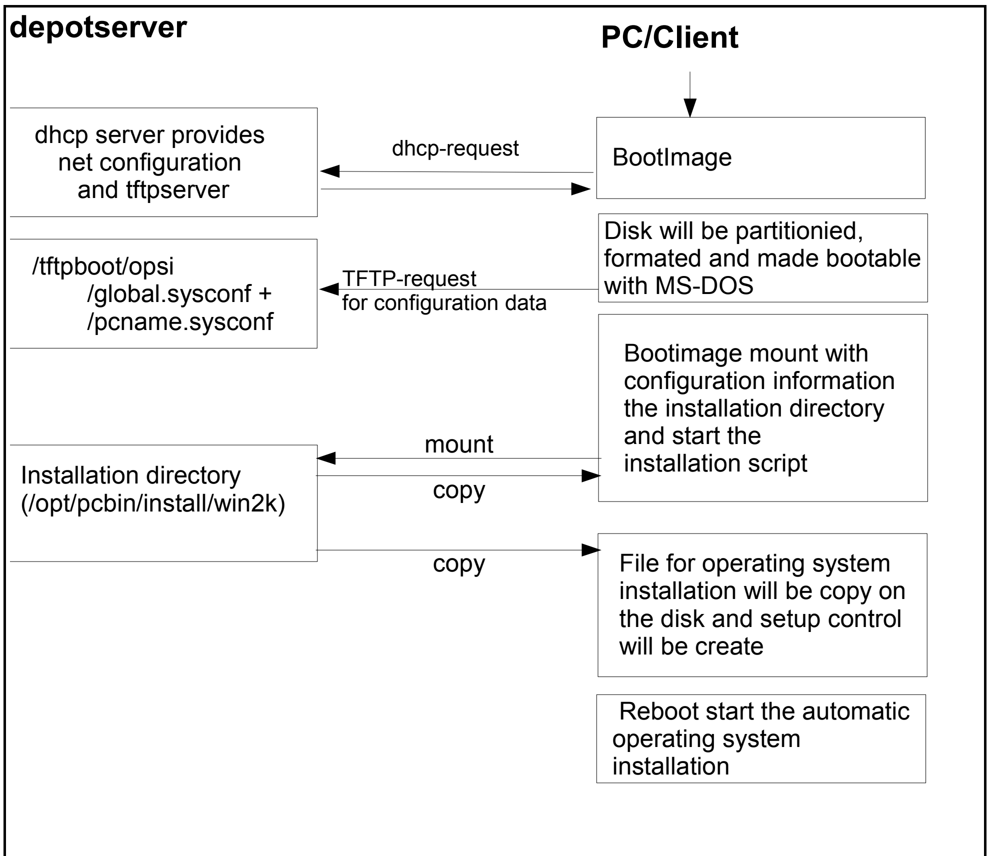
According to different situations, several information has to be provided for the CD boot image by interactive input. The most simple case is to provide no further information. Eventually the clients hostname can be passed by hn=<hostname>. Using the option ASK_CONF=1 several parameters can be queried.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

5.1.5. 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 files global.sysconf, <domain>.sysconf and <pcname>.sysconf will be loaded from the directory /tftpboot/opsi.

The configuration file 'global.sysconf' provides for all clients and every installable OS the information about the server in charge, the file share and the name of the installation script.

The file <pcname>.sysconf 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. Exact informations about configuration and content you will find in chapter 10 'Important files of a opsi depot server'.


depotserver
PC/Client
dhcp server provides net configuration and tftpserver
dhcp-request
BootImage
Disk will be partitionied, formated and made bootable with MS-DOS
/tftpboot/opsi /global.sysconf + /pcname.sysconf
TFTP-request for configuration data
Bootimage mount with configuration information the installation directory and start the installation script
mount
copy
Installation directory (/opt/pcbin/install/win2k)
copy
File for operating system installation will be copy on the disk and setup control will be create
Reboot start the automatic operating system installation

Figur 23: PXE-Boot loaded with bootimage preparing hard disk for operating system installation

These informations will be collected in a way that specific information taken from <pcname>.sysconf overwrites the more general information from global.sysconf. This will be combined with some information taken from the dhcp response (e.g. which is the tftp-server) 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 2000 installation will be described.

Prepare the disc: On the hard disk the bootimage creates a new partition (of size 2 - 6 GB), formats it and installs a bootable DOS 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/win2k/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: An 'autoexec.bat' file will be created, which will start the Windows 2000 setup program at the next reboot. The patched 'unattend.txt' is passed to the setup as the control file for unattended installation.

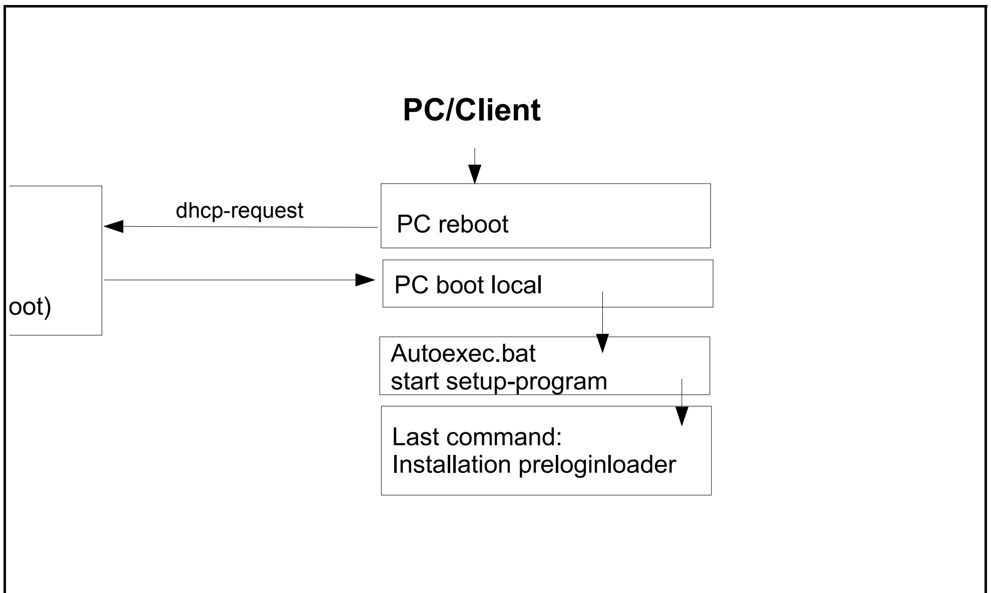


Figure 24: After preparation of the bootimage the computer starts from local disk and installs the operating system and the opsi-PreLoginLoader

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 MS-DOS will be started and the 'autoexec.bat' starts the setup for operating system installation.

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).

5.1.6. 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 **”on”** 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.

5.1.7. How the patcha program works

As mentioned above the information collected from dhcp, global.sysconf and pcname.sysconf 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: patcha 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#>
```

5.1.8. Integrating additional drivers in the unattended Windows installation

If some client hardware isn't supported by the standard Windows drivers, it could be useful and sometimes even necessary to integrate new drivers into the unattended installation. Regarding network devices this is very recommendable, because a client without network is neither remote accessible, nor can the automated software distribution connect to any distribution file shares.

The server can provide additional drivers to be installed during Windows setup. All of these drivers must come with an '.inf' file, which holds the driver's installation information. Any drivers which are packed as an executable cannot be used for this (but often they can be unpacked to get the plain installation files).

If you just have some differing hardware, you can take drivers as provided by the hardware manufacturer and put this to the distribution file share.

But if you have to support a lot of different hardware, it might be convenient to use ready packed packages for a whole bunch of Windows XP-drivers as provided by http://driverpacks.net/.

5.1.8.1. Simplified driver integration with symlinks

This method requires at least opsi V3.1 and winxppro_sp2-3.opsi.

The script '`download_driver_pack.py'` is part of the winxppro opsi package. It downloads the current driverpacks (!!! about 2,5 GB !!!) from http://driverpacks.net/DriverPacks/overview.php to the winxppro directory.

Additional drivers can be added, each into its own subdirectory of **`winxppro/drivers/drivers/preferred`** .

Then execute the script '**`create_driver_links.py'`** from the winxppro directory. The script searches all the directories beneath 'drivers' and creates links to assign drivers to its hardware. Drivers found beneath the directory '**`preferred'`** will be taken as preferential. During installation the script '**`winxxpro.py`**' from the bootimage identifies the client hardware and the required drivers. These will be copied onto the hard disk and the 'unattend.txt' (which is the control file for unattended installation) will be patched according to this.

5.1.8.2. Driver integration classic

The classic way to add special drivers to the unattended installation is to put them into the OEM directory. This is required:

- An .inf-file for each driver
- One directory per driver beneath the OEM directory
- Path to these directories in the unattend.txt

For each additional driver there must be an '.inf' file holding the installation information and a set of drivers (usually '.dll', '.sys' etc.). If you just have an executable ('.exe'), it often can be unpacked to get the plain installation files. But without '.inf' file a driver cannot be added to unattended setup the classic way.

This inf file must be copied together with the other installation files to a subdirectory of OEM\$1.

Here is an example for adding drivers for Realtek 8139 network controllers:
Copy the drivers (including '.inf') to

```
P:\install\winxppro\$oem$\$1\setup\nic\rtl8139
```

In this case the opsi depot server share for 'Windows XP Professional' is 'P:\install\winxppro'.

In order to inform the Windows setup program where to look for additional drivers you have to patch the „OemPnpDriversPath“ entry of 'unattend.txt'. It must contain every driver path to search (relative to $1) separated by semicolons, e.g:

```
OemPnpDriversPath="setup\vga\ati;setup\nic\rtl8139"
```

Remember:

The names of all driver files have to follow the 8.3 file naming convention, which is up to 8 characters for the filename and up to 3 characters for the file extension.

Another restriction is the limited string length for 'OemPnpDriversPath', which is up to 99 chars for Windows NT and older versions of Windows 2000. For an updated Windows 2000 it is limited to 255 chars and 4096 chars for Windows XP. So it is always a good idea to keep directory names rather short and within a flat hierarchy.

Additional information

It is especially important to include the drivers for special network controllers to the unattended installation, because a working network is required to get the software distribution system running. Almost any other driver (graphic, sound ...) could also be installed later on by the software distribution system.

On the internet you can find a lot of information on driver integration, for instance at: http://www.windows-unattended.de

5.2. 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.

5.3. Memtest

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

5.4. 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).

6. opsi-Module: depot server

6.1. 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 scriptable 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).

6.2. Installation and initial operation

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

6.3. 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/

6.4. Shares for software packets and configuration files

6.4.1. 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 configuration file '/tftpboot/opsi/global.sysconf'. 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.

6.4.2. Required administrative user accounts and groups

6.4.2.1. User opsiconfd

The opsiconfd deamon is started as user 'opsiconfd'.

6.4.2.2. 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.

6.4.2.3. 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`**'.

6.4.2.4. 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`**'.

6.4.3. 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.

6.4.4. 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 are located in '/var/lib/opsi/config'.

6.4.5. 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.

6.5. Administration of PCs via DHCP

6.5.1. 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).

6.5.2. 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'.

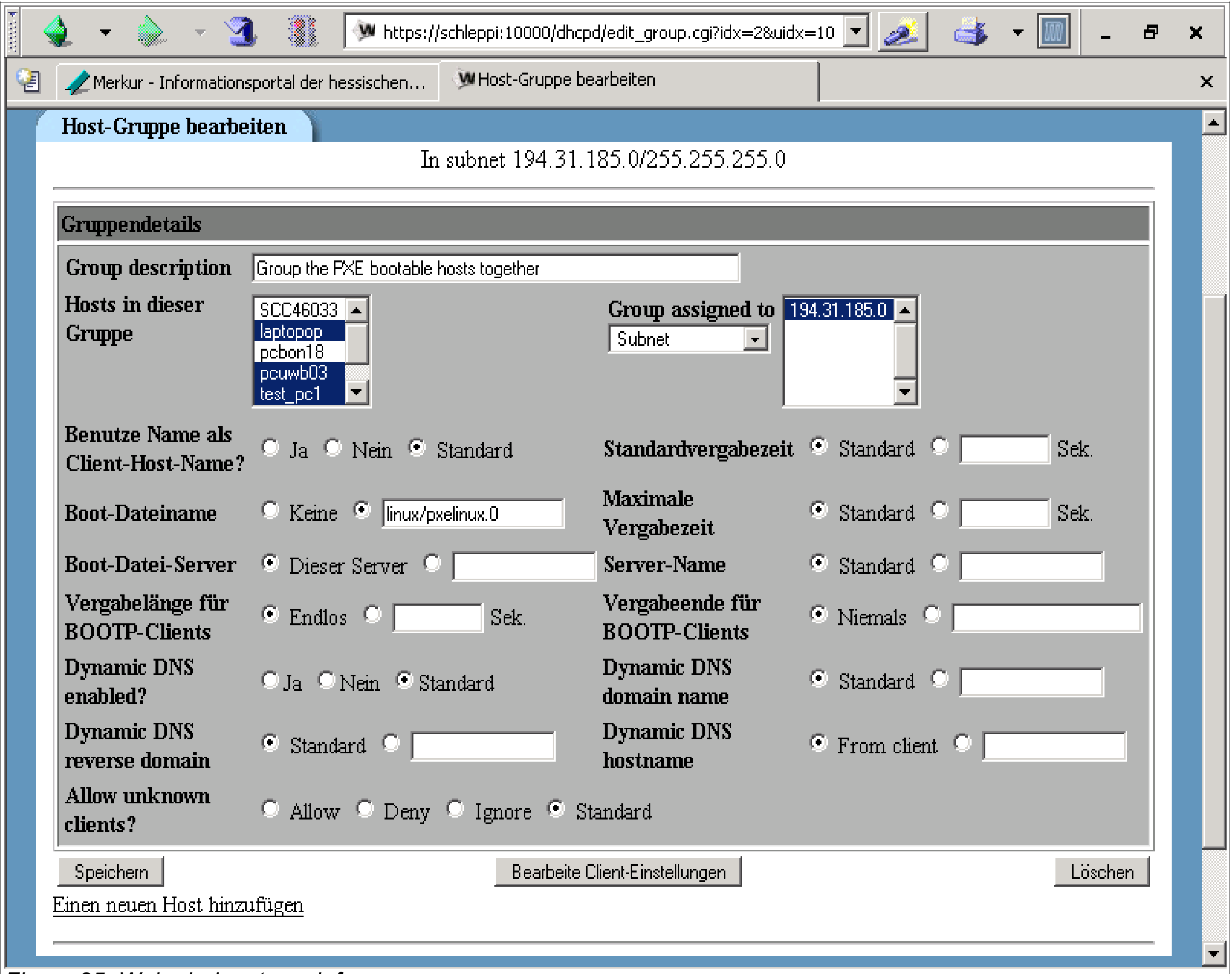
If a new PC should join the subnet, it has to be registered in 'dhcpd.conf'.

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

6.5.3. 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'`.


https://schleppi:10000/dhcpd/edit_group.cgi?idx=2&uidx=10

Merkur - Informationsportal der hessischen… | Host-Gruppe bearbeiten

Host-Gruppe bearbeiten

In subnet 194.31.185.0/255.255.255.0

Gruppendetails

Group description	Group the PXE bootable hosts together		
Hosts in dieser Gruppe	SCC46033, laptopop, pcbon18, pcuwb03, test_pc1	**Group assigned to** Subnet	194.31.185.0
Benutze Name als Client-Host-Name?	Ja Nein [✓ Standard]	**Standardvergabezeit**	[✓ Standard] ○ Sek.
Boot-Dateiname	Keine [✓ linux/pxelinux.0]	**Maximale Vergabezeit**	[✓ Standard] ○ Sek.
Boot-Datei-Server	[✓ Dieser Server] ○	**Server-Name**	[✓ Standard] ○
Vergabelänge für BOOTP-Clients	[✓ Endlos] ○ Sek.	**Vergabeende für BOOTP-Clients**	[✓ Niemals] ○
Dynamic DNS enabled?	Ja Nein [✓ Standard]	**Dynamic DNS domain name**	[✓ Standard] ○
Dynamic DNS reverse domain	[✓ Standard] ○	**Dynamic DNS hostname**	[✓ From client] ○
Allow unknown clients?	Allow Deny Ignore [✓ Standard]		

Speichern | Bearbeite Client-Einstellungen | Löschen

Einen neuen Host hinzufügen

Figure 25: 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).

6.6. 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.

7. opsi data storage (backend)

7.1. File backend

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

7.1.1. 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”.

7.1.2. File-Backend (opsi 3.0)

Basic features of the backend 'File' :

- deprecated and shouldn't be used with new installations
- not Linux standard base conform
- backward compatible to opsi 2.x/3.0
- not all of the opsi functions are available with this backend.

- works also with clients in 'Classic' mode (clients that do not get their configuration information from the opsi service, but by direct access to the configuration files).

The configuration files are spread to the following areas.

- Configuration data files in the tftp area
  In the directory '/tftpboot/opsi' are the general client configuration file 'global.sysconf' and the overriding <pcname>.sysconf files with the client specific configurations.

- Config share
  in the directory '/opt/pcbin/pcpatch' are the client software configuration files and depot configurations. This area is shared as config share per Samba.

- Logging and hard- and software inventory
  The hardware informations collected from the boot image will be saved as '<configshare>/pclog/<pcname>.hw'.
  The software inventory information detected by the product 'softinventory' will be saved as '<configshare>/pclog/<pcname>.softinv'.

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

<u>7.2. LDAP backend</u>

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.

7.2.1. 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 }
```

7.2.2. Configuring the LDAP-backend

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

7.2.3. Assign the LDAP-backend to methods

```
    self.defaultBackend = BACKEND_LDAP
    self.bootimageBackend = BACKEND_REINSTMGR
    self.passwordBackend = BACKEND_FILE
    self.pckeyBackend = BACKEND_FILE
```

In this example the LDAP backend is set as the default backend. The PC-keys and the pcpatch-passwords are still administrated as files ('/etc/opsi/pckeys' and '/tftpboot/opsi/<pcname>.sysconf').

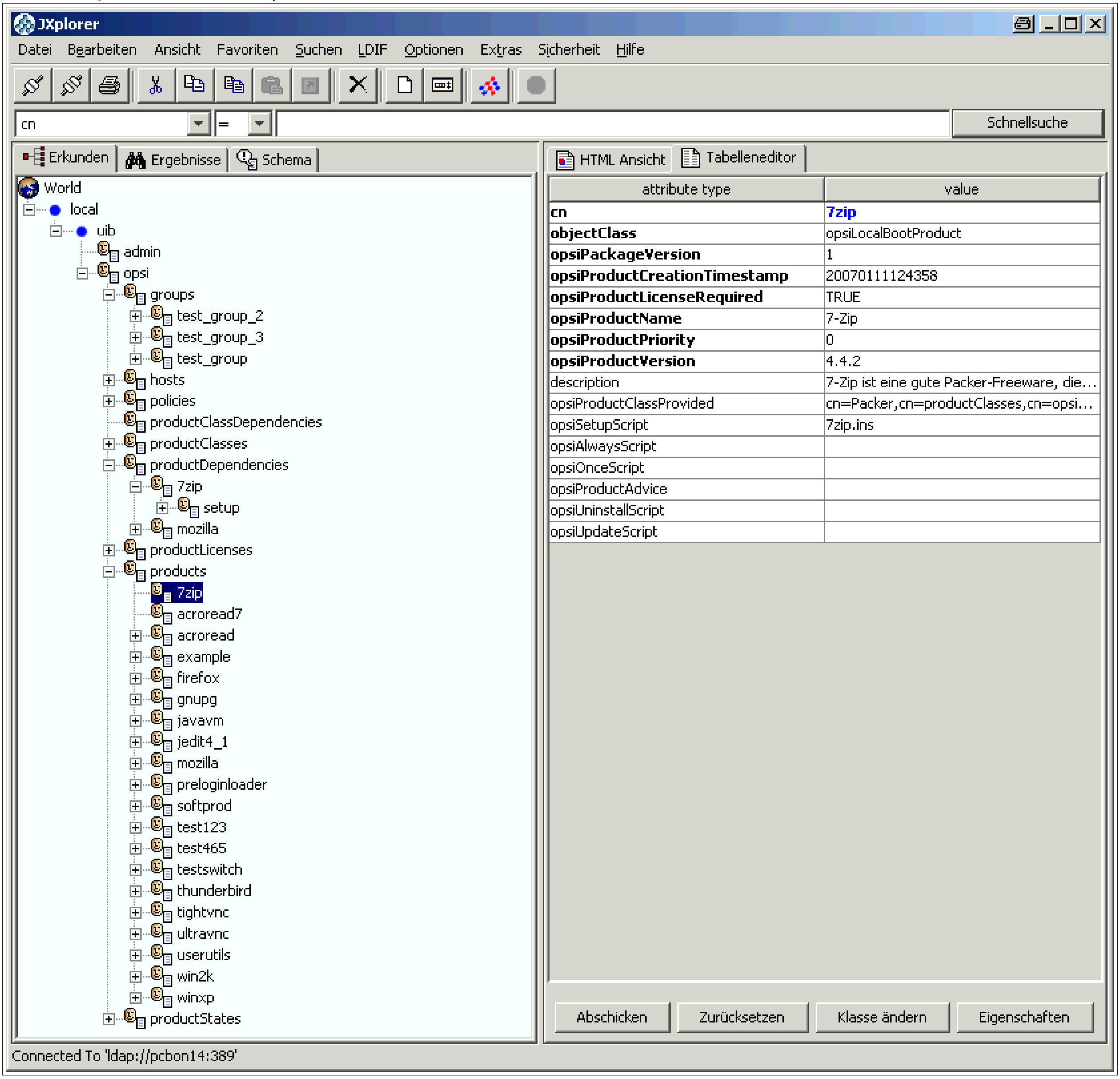
Now restart the opsi config daemon 'opsiconfd':

```
/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).



attribute type	value
cn	**7zip**
objectClass	opsiLocalBootProduct
opsiPackageVersion	1
opsiProductCreationTimestamp	20070111124358
opsiProductLicenseRequired	TRUE
opsiProductName	7-Zip
opsiProductPriority	0
opsiProductVersion	4.4.2
description	7-Zip ist eine gute Packer-Freeware, die...
opsiProductClassProvided	cn=Packer,cn=productClasses,cn=opsi...
opsiSetupScript	7zip.ins
opsiAlwaysScript	
opsiOnceScript	
opsiProductAdvice	
opsiUninstallScript	
opsiUpdateScript	

Connected To 'ldap://pcbon14:389'

7.3. 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.

7.4. Boot files

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

7.5. 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 authentification 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 is stored in '/tftpboot/opsi/<pcname>.sysconf' and will be transferred to the client at boot time. Hence the servers 'pcpatch' password can be changed any time. The new encrypted password will be sent to every client at the next reboot.

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

wInst as part of the opsi preloginLoader is the program which executes the wInst-scripts and installs the software packets on the client according to the clients configuration data. During the product installation the installation progress is shown full screen on the clients display. You can configure the look with the file
'<utils share>**`\utils\wInst_p.ini`**'
to change the logo or the colors to integrate your corporate identity (CI).

The typical '**`wInst_p.ini`**' looks like this:

```
picture1=wInst2.bmp
label1="uib gmbh"
picture2=wInst1.bmp
label2="opsi - Open Pc ServerIntegration"
;available colors: blue, aqua, green, lime, maroon, navy, teal, white, yellow,
olive; default: blue; (or color code)
backgroundcolor=
textcolor=
```

Also the 'netmount' start window can be adapted, which is shown while connecting the file shares, before the software installation starts. You can change the registry values of '**`bitmap1`**' and '**`bitmap2`**' in the registry key '**`HKEY_LOCAL_MACHINE\SOFTWARE\opsi.org\pcptch`**' to your preferred logo. The labels for the logo can be set by '**`label1`**' and '**`label2`**' in the same registry key. If no logos should be shown, set the registry value '**`loadbitmap`**' to '0'. The variable '**`backgroundcolor`**' is for adapting the background color of the window. The value of '**`textcolor`**' sets the foreground color (text color). You find the available colors in the example above (taken from '**`wInst_p.ini`**').

Those registry values can also be managed with the '**`global.sysconf`**' on the server or with '**`<pcname>.sysconf`**' for individual PCs. Thereto you have to set the corresponding values in the section **`[pcptch]`**. The effects of these changes (for technical reasons) are visible after the second re-boot of the client.

9. Overview: A PC boots from the network

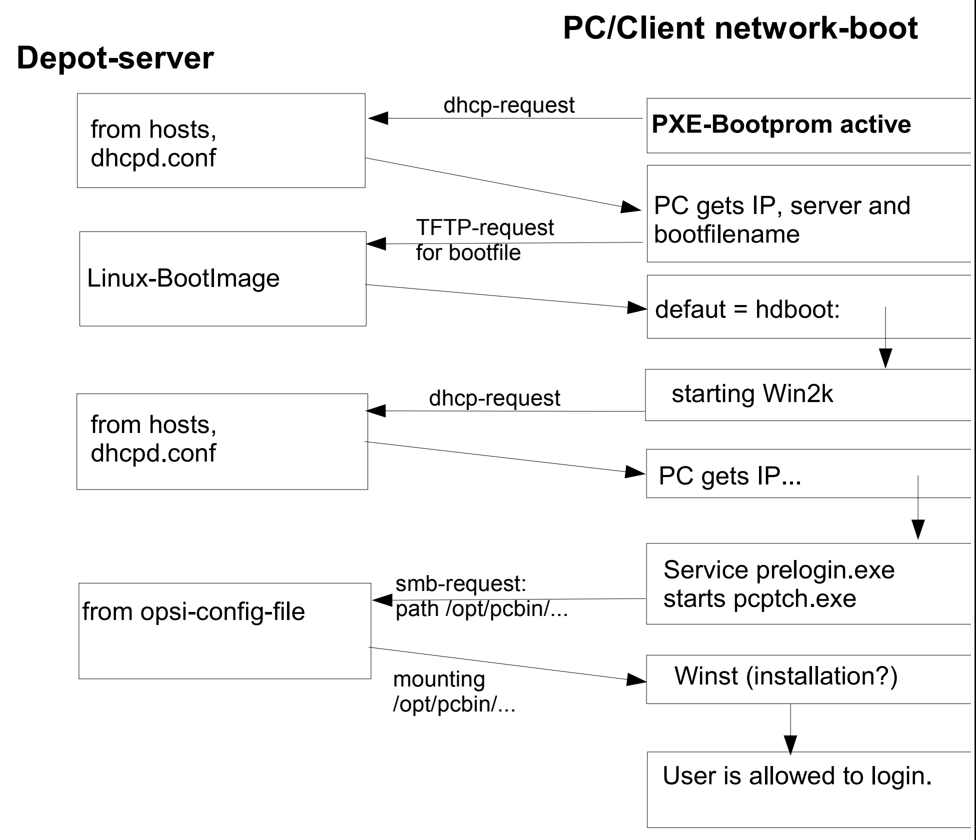


Figure 27: Flowchart for a 'regular' PXE-boot without re-installation, but with the start of the opsi preLoginLoader

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.

10. Important files on the depot servers

10.1. Configuration files

10.1.1. Configuration files in /etc

10.1.1.1. /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', inclusing 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
```

10.1.1.2. /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).

10.1.1.3. /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
```

10.1.1.4. /etc/opsi/passwd

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

10.1.1.5. /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.

10.1.1.6. /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).

10.1.1.7. /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.

10.1.1.8. /etc/opsi/opsiconfd.conf

Since opsi V3

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

10.1.1.9. /etc/opsi/opsiconfd.pem

Since opsi version 3.0

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

10.1.1.10. /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.

10.1.1.11. /etc/opsi/version

Holds the version number of the installed opsi.

10.1.1.12. /etc/init.d/

Start and stop scripts for

- opsi-atftpd
- v3 opsiconfd
- v3.1 opsipxeconfd

10.2. Boot files

10.2.1. Boot files in /tftpboot/linux

10.2.1.1. pxelinux.0

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

10.2.1.2. install und miniroot.gz

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

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

10.2.2.1. 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.

10.2.2.2. default

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

10.2.2.3. install

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

10.3. 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.

10.3.1. File3.1-Backend

10.3.1.1. 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.

```
/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'.

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

10.3.1.2.1. clientgroups.ini

This file holds information on the client-groups.

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

Example

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

10.3.1.2.2. 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'.

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

10.3.1.3.1. <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:

10.3.1.3.1.1. [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

10.3.1.3.1.2. [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

10.3.1.3.1.3. [localboot_product_states]

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

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

e.g.

```
firefox = installed:setup
```

10.3.1.3.1.4. [netboot_product_states]

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

e.g.

```
winxppro = installed:none
```

10.3.1.3.1.5. [<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
```

10.3.1.3.1.6. [<product>-install]

```
product property = value
```

e.g.

```
viewer = off
```

10.3.1.3.1.7. [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
```

10.3.1.4. 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.

10.3.1.5. 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
```

10.3.1.6. 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.

10.3.2. File-Backend (opsi 2.x/3.0)

This backend is 'deprecated' and only for backward compatibility.

10.3.2.1. Configuration files in /tftpboot/opsi

Since opsi-version 2.5 entries in the '*.sysconf'-files are case sensitive.

10.3.2.1.1. *.sysconf-files

The '*.sysconf'-files contain information which is applied during the OS-installation and/or at client startup. The internal file structure is like 'ini'-files.

The content of the sections [general] and [shareinfo] is retrieved by the bootimage and saved on the client as the file 'sysconf.ini' in the 'cfg'-directory of the 'Preloginloader'. The data are also written to the file 'patcha.ini', which is used for patching other configuration files.

10.3.2.1.2. global.sysconf

```
[General]
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 smb is supported.
2. Share name (\\laptop\opt_pcbin): This is the share to be mounted. In case of a drive letter given further down in this file, the share is mounted as this drive letter.
3. The path, in which the software installation packages are stored.

```
depotdrive=<driveletter the depoturl is mounted to>
```

Example: P: (including the colon)

```
configdrive=<driveletter the configurl is mounted to >
```

Example: P: (including the colon)

```
utilsdrive=<driveletter the utilsurl is mounted to >
```

Example: P: (including the colon)

```
OS=Win2k
```

OS to be installed by default.

```
[shareinfo]
```

```
pcpatchpass=a788f2614d04ddd435e08418ec97e130
```

Encrypted password for the user 'pcpatch' (only available if there is a value for 'global' in '/etc/pckeys'). Will be overridden by an entry in the 'pcname.sysconf'.

Product sections, as an example for win2k:

```
[Win2k]
```

Informations about the client OS Windows 2000

```
extendoem=10240
```

Size (MB) of the NTFS partition to be created.
0=keep old size. 1=use all of the available space.

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

URL referring to the directory containing the installation and config files.
Example: `os-instpath=smb://schleppi/opt_pcbin/install/win2k`

```
instscript=win2k.py
```

OS Installation-script (executed by the boot image for this OS).

```
askBeforeInst=false
```

Defines whether the bootimage waits for reinstallation approval by the user or starts the installation without questioning. Is this entry missing or anything else than 'false' , '0' or 'no', a confirmation request is executed. This entry since opsi V2.5 is in the product section (it was expected to be in the [general] section before).

```
[WinXP]
```

The next section holds the configuration information for WinXP.

10.3.2.1.3. domain.sysconf

For future use (will store domain specific information).

10.3.2.1.4. <pcname>.sysconf

Since opsi-version 2.5 no definitions regarding the operating system are allowed in this file. It has to be written to the 'global.sysconf'.

```
[shareinfo]
pcpatchpass=a788f2614d04ddd435e08418ec97e130
```

Encrypted password for the user 'pcpatch' (for connecting the installation share).

```
[General]
OS=Win2k
```

OS to be installed on this client.

```
[Win2k]
extendoem=10240
.....
```

Information for Windows 2000

```
[WinXP]
ProductKey=0815123
.....
```

Information for Windows XP

10.3.2.2. Configuration files in the opsi config and utils file shares

10.3.2.2.1. Function and configuration of <pcname>.ini files

For every client computer exists a configuration file to control the software distribution. These files are stored at the configuration share (per default this is '/opt/pcbin/) of the opsi depot servers in the subdirectory 'pcpatch'. The name of the configuration file is the IP-name of the computer with the extension '.ini'.

Example:
/opt/pcbin/pcpatch/pcmueller.ini.

These text files have the structure of INI-files. The section [products-installed] lists all the software packets which are available at the opsi depot server. Each software packets has a switch setting which represents the status of that software on the client.

Example:
[products-installed]
mozilla=off
firefox=on

The **switch settings** for every product have the following meaning:

setup:	Product will be installed at the next boot (then the switch is set to 'on')
on:	Product is installed - no action required
off:	Product isn't installed on the computer - no action required
deinstall:	Product will be deinstalled at the next boot (then switch to 'off')
update	Product will be updated at the next boot (then switch to 'on')
always	Product install script will execute at every boot (stays as 'always')

once	Product install script will execute one time only (and than switch to 'off') (whereas 'setup' will switch to status 'on')

At every boot the switch setting will be checked by the program wInst. If there is an action request, the appropriate script will be executed to install, uninstall or update the software packet and then the switch is set to the resulting status.

If there is no config file for a PC it will be created from the prototype file 'pcproto.ini'. So you should configure the 'pcproto.ini' to have the standard software packets set to 'on'.

To change these files you need a Unix user account at the opsi depot server and you should be member of the unix group 'pcpatch'.

Example for a <pcname>.ini file:

```
[Products-installed]
acroread=on
virdat=on
mozilla=on
perl=on
virscan=on
javavm=on
citrix_c=off
ooffice=on
integtools=off
tightvnc=off
jedit4_1=off

[mozilla-install]
disable_ntlm=off
prefbar=on
calendar=on
open_eml=on
```

v3 The client information from the opsi-configurator is saved in the section [info]. Also the last client connect to the opsiconfd will be saved.

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

v3.1 In the section [netbootproducts-states] the status and action request for netboot products will be saved the way described for the File31 backend.

10.3.2.2.2. Software-product-information file: produkte.txt

Example:

```
; Filename: produkte.txt
; lists product-dependencies (product-requires-Sections,
; product-requires_before-sections, product-requires_after-sections)
; short descriptions of products (produkt-info-sections)

; Some of the keywords are currently in German
; Produktname -> product name
; Infotext    -> info text
; Hinweis     -> additional advice

[acroread-info]
Produktname=Acrobat Reader 5.1
Infotext=for mozilla
Hinweis=removes Acrobat Reader V3.4

[virscan-requires_before]
nt4sp6a=on

[virdat-info]
Produktname=virdat: Actual virus signature files
Infotext=signature files V4

[mozilla-info]
Produktname=Mozilla 1.6
Infotext=Mozilla 1.6 including customising tool
Hinweis=Additional switches for calendar, prefsbar and handling of .eml-files

[perl-requires]
mozilla=on

[perl-info]
Produktname=ActivePerl from ActiveState
Infotext=ActivePerl 5.6.1.631 MSWin 32
Hinweis=

[virscan-info]
Produktname=virscan: Virenscanner V7.1.0
Infotext=Network Associates VirusScan v7.1.0
Hints=The actual signature files are in the product virdat

[javavm-info]
Produktname=1.3.1_07 / 1.4.1_02
Infotext=This product installs two different JavaVM
Hints=The switch default13=on (default) installs javavm 1.3 as default

[citrix_c-info]
Produktname=Citrix client 6.31.1051(128-Bit SSL)
Infotext=Terminal client for Citrix WTS

[mozilla-requires]
hupsutil=on

[mozilla-requires_after]
javavm=setup
acroread=setup

[ooffice-info]
Produktname=OpenOffice 1.1.1
Infotext=
Hinweis=
[ooffice-requires_before]
javavm=on

[integtools-info]
Produktname=integtools: integrationstools V.6.12.2003
Info text=some tools to create wInst scripts
Hinweis=

[jedit4_1-info]
Produktname=jEdit 4.1
Infotext=Open Source Editor (Java) with syntax highlighting and plugins
Hinweis=Java Engine at least Version 1.4 recommended

[jedit4_1-requires]
javavm=on

[jedit4_1-requires_before]
javavm=on

[tightvnc-info]
Produktname=tightvnc 1.2.9: remote control of computers (compressed transfer)
Infotext=Standard installation
Hinweis=GUI English

[virscan-requires_after]
virdat=setup

[virdat-requires_before]
virscan=on

[ie6-requires_before]
nt4sp6a=on

[vobsutil-requires_before]
javavm=on
```

10.3.2.2.3. Software-product-path and script-information: pathnams.ini

Example 'pathnams.ini':

```
;Remark: This file contains all general product related entries
;         (which are not specific for a single PC)
;      In combination with <pcname>.ini this will configure the PC

[acroread-install]
setupPath=p:\install\acroread
setupWinst=acroread.ins

[virdat-install]
setupPath=p:\install\virdat
setupWinst=virdat.ins

[vnc-install]
setupPath=p:\install\vnc
setupWinst=vnc.ins

[mozilla-install]
```

```
setupPath=P:\install\mozilla
setupWinst=mozilla.ins
deinstallWinst=delmoz.ins

[perl-install]
setupPath=p:\install\perl
setupWinst=perl.ins

[virscan-install]
setupPath=p:\install\virscan
setupWinst=virscan.ins
```

(....)

10.3.2.3. Help files produkte.txt and pathnams.ini

Apart from the client specific software configuration file ('<pcname>.ini') there are two other configuration files with information regarding the installation of software packets. These files are patched automatically by installing opsi packets on the server. Usually the administrator will not have to edit them.

In the file 'pathnams.ini' (default is '/opt/pcbin/pcpatch/pathnams.ini') for every product is defined where the program wInst will find the installation script. If there is also an uninstall or an update script, the script path is written to this file.

In the file 'produkte.txt' (default is '/opt/pcbin/utils/produkte.txt') are important additional product informations saved. You can see those informations from the ini-editor. You also see the information about the product dependencies. Product dependencies are evaluated bei wInst and affect the installation order of products. The dependency types are:

- Product A 'requires' product B:
  the operation of product A is based on product B. So, if product A is to be installed, product B must be installed also.
- Product A 'requires_before' product B:
  already the installation of product A requires the presence of product B. So product B has to be installed before the installation of product A can be started.
- Product A 'requires_after' the installation of product B:
  Even if product B is already installed, it has to be installed (again) after the

installation of product A is completed (for instance product B installs plugins for product A).

The detailed syntax of this file is described in the 'opsi integration handbook'.

10.4. Files of the LDAP-backend

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

10.5. Opsi programs and libraries

10.5.1. Python library

The opsi python modules are located at:

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

or

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

10.5.2. 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.

10.5.3. Programs in /usr/bin

- v3 opsi-admin
  Starts the command line interface for the opsi python library
- v3.1 opsi-configed
  Command to start the opsi-management interface
- v3.1 opsi-convert
  Script for converting between different backends.

- v3.1 opsideinst
  Script for deleting products
- v3 opsiinst (opsiinstv2)
  Script to unpack and install opsi packets on the server
- v3.1 opsi-makeproductfile (opsi-makeproductfilev2)
  Script for packing the opsi-packet (opsiV2 compatible Version)
- v3 opsi-newprod
  Script for creating a new opsi product
- v3 makeproductfile (makeproductfilev2) (deprecated)
  Replaced by opsi-makeproductfile
  Script for packing the opsi-packet (opsiV2 compatible Version)
- v3 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

10.6. opsi-log files

10.6.1. /var/log

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

10.6.2. /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.

10.6.3. /var/log/opsi/opsipxeconfd

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

10.6.4. OS-installation

The boot image logs its actions to '<config share>/pclog/<pcname>.bi'. If the boot image couldn't connect the config share, the logs are written to '/tmp/log'.

10.6.5. 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.

11. Registry entries

11.1. Registry entries for the opsi-preLoginLoader

11.1.1. 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>
```

11.1.2. 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')

11.1.3. 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.

v3 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

v3 opsiServiceURL
The URL to connect the opsi service
e.g. https://bonifax.uib.local:4447

v3 repeatServiceConnectNo
Number of retries for connecting the service (default 3).

11.2. Registry-entries for opsi-wInst

11.2.1. 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)
```

12. 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).

	Dependant 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 deamon - 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.

13. Table of Figures

Table of Figures

14. Additions and Changes

Additions and changes in this hand book.

14.1. 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

14.2. 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)

14.3. Additions opsi 2.5 / opsi 3.0 (12/8/06)

- Registry entry 'button_stopnetworking' is in 'opsi.org/pcptch'

14.4. 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

14.5. Additions opsi 3.0

v3 12.4.07: LDAP chapter

14.6. 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

14.7. Additions opsi 3.2 (21.11.07)

Actualized: the chapter 'Simplified driver integration with symlinks' for driver integration (**`download_driver_pack.py`** and **`preferred`**)